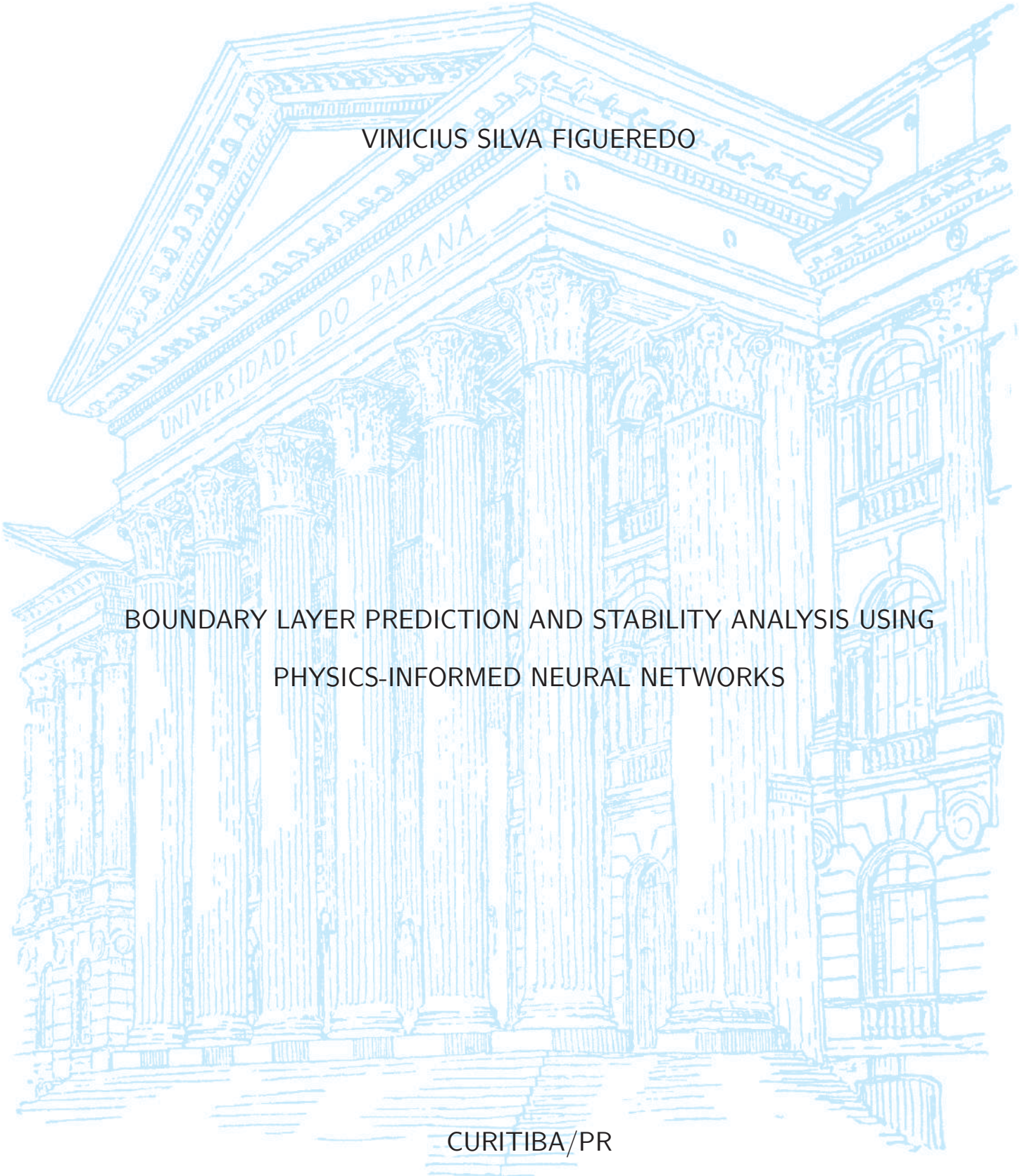
UNIVERSIDADE FEDERAL DO PARANÁ

VINICIUS SILVA FIGUEREDO



BOUNDARY LAYER PREDICTION AND STABILITY ANALYSIS USING PHYSICS-INFORMED NEURAL NETWORKS

CURITIBA/PR

2025

VINICIUS SILVA FIGUEREDO

BOUNDARY LAYER PREDICTION AND STABILITY ANALYSIS USING PHYSICS-INFORMED NEURAL NETWORKS

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal do Paraná, como requisito parcial para a obtenção do título de Mestre em Ciências da Engenharia Mecânica.

Orientador: Prof. Dr. Luciano Kiyoshi Araki

Coorientador: Prof. Dr. Gustavo Luiz Olichevis Halila

CURITIBA/PR

2025

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

Figueredo, Vinicius Silva
Boundary layer prediction and stability analysis using physics-informed neural networks / Vinicius Silva Figueredo. – Curitiba, 2025.
1 recurso on-line : PDF.

Dissertação (Mestrado) - Universidade Federal do Paraná, Setor de Tecnologia, Programa de Pós-Graduação em Engenharia Mecânica.

Orientador: Luciano Kiyoshi Araki
Coorientador: Gustavo Luiz Olichevis Halila

1. Redes neurais (Física). 2. Camada limite laminar. 3. Escoamento. 4. Modelagem física. I. Universidade Federal do Paraná. II. Programa de Pós-Graduação em Engenharia Mecânica. III. Araki, Luciano Kiyoshi. IV. Halila, Gustavo Luiz Olichevis. V. Título.

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação ENGENHARIA MECÂNICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **VINICIUS SILVA FIGUEREDO**, intitulada: **BOUNDARY LAYER PREDICTION AND STABILITY ANALYSIS USING PHYSICS-INFORMED NEURAL NETWORKS**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 12 de Novembro de 2025.

Assinatura Eletrônica
13/11/2025 12:27:57.0
GUSTAVO LUIZ OLICHEVIS HALILA
Presidente da Banca Examinadora

Assinatura Eletrônica
15/11/2025 16:05:56.0
JOSÉ MESSIAS MAGALHÃES JÚNIOR
Avaliador Externo (GEORGIA INSTITUTE OF TECHNOLOGY)

Assinatura Eletrônica
14/11/2025 00:48:05.0
JOAO LUIZ FILGUEIRAS DE AZEVEDO
Avaliador Externo (INSTITUTO DE AERONáUTICA E ESPAçO)

To my parents, for their unconditional support and for always encouraging me to pursue my dreams. To Raquel, for her constant love, care, and affection, which made this journey lighter and more meaningful.

ACKNOWLEDGMENTS

I would first like to thank my parents for their continuous support and for always encouraging me to give my best. To my partner Raquel, for her affection, understanding, and companionship throughout this journey, whose support made everything possible. I am also grateful to my aunt and uncle, Lêda and Otacílio, for their constant encouragement and assistance.

I would like to express my gratitude to my advisors, Prof. Dr. Gustavo Halila and Prof. Dr. Luciano Araki, for their patience, guidance, and valuable contributions during the development of this work. In particular, I thank Prof. Dr. Gustavo Halila for his dedication and generosity in sharing his knowledge.

I am also thankful to Prof. Dr. José M. Magalhães Júnior for his support throughout this research and for his teachings, which helped me achieve the results presented here.

I would like to give special thanks to Prof. Dr. Nuccia Carla Arruda de Sousa, who introduced me to the path of scientific research and has become a major reference in my academic and professional life.

This work was financially supported by the Coordination for the Improvement of Higher Education Personnel (CAPES), to whom I extend my sincere thanks.

RESUMO

A modelagem de escoamentos baseada em dados e informada por física oferece uma abordagem eficiente e facilmente automatizável para a simulação de diversos problemas em mecânica dos fluidos. Tais modelos podem ser facilmente integrados a outros códigos para solução de escoamentos, constituindo uma alternativa prática às redes neurais padrão utilizadas como substitutas em simulações de dinâmica dos fluidos computacional (CFD). Enquanto redes neurais convencionais exigem grandes volumes de dados e elevado tempo computacional para o treinamento, além de poderem produzir erros inaceitáveis em certos cenários , as redes neurais informadas por física (PINNs) aproveitam as leis físicas subjacentes, representadas por equações diferenciais parciais, para acelerar o treinamento e aprimorar a acurácia da modelagem. Neste trabalho, as PINNs são aplicadas ao problema da camada limite laminar sobre uma placa plana, recuperando com sucesso métricas fundamentais, como perfis de velocidade e a evolução espacial da espessura da camada limite. A abordagem é também estendida para investigar a evolução espacial de modos de perturbação sobrepostos a uma camada limite laminar. Um código de estabilidade de escoamentos baseado nas equações de estabilidade parabolizadas (PSE) é utilizado para gerar os dados empregados nessas investigações de estabilidade espacial. Os resultados correspondentes mostram que abordagens baseadas em dados e informadas por física podem representar, de forma confiável, fenômenos oscilatórios em escoamentos limitados por paredes, fornecendo previsões acuradas dos modos de estabilidade em uma formulação não local e não paralela de análise de estabilidade de escoamentos.

Palavras-chave: Rede neural informada por física, Camada limite laminar, Escoamento limitado por parede, Modelagem orientada por dados.

ABSTRACT

Physics-informed data-driven modeling of fluid flows provides an efficient and easily automatable approach to simulate a variety of fluid mechanics problems. Such models can be seamlessly integrated with other flow solvers, offering a practical alternative to standard neural networks as surrogates for computational fluid dynamics (CFD) simulations. While conventional neural networks require large volumes of data and computational time for training and may produce unacceptable errors in certain scenarios, physics-informed neural networks (PINNs) leverage the underlying flow physics, represented by partial differential equations, to accelerate training and enhance modeling accuracy. In this work, PINNs are applied to a laminar flat plate boundary layer problem, successfully recovering key metrics such as velocity profiles and the boundary layer thickness evolution. The approach is further extended to investigate the spatial evolution of boundary layer disturbance modes. A flow stability solver based on the parabolized stability equations (PSE) is used to generate the data for the spatial stability investigations. The corresponding results show that physics-informed, data-driven approaches can reliably represent wave-like phenomena in wall-bounded flows, providing accurate predictions of flow stability modes in a nonlocal, nonparallel flow stability framework.

Keywords: Physics-informed neural networks, Laminar boundary layer, Wall-bounded flows, Data-driven modeling.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ADADELTA	Per-dimension learning rate method for gradient descent
ADAGRAD	Adaptive gradient optimizer
ADAM	Adaptive moment estimation
BVP	Boundary value problem
CFD	Computational fluid dynamics
CNNs	Convolutional neural networks
DNS	Direct numerical simulation
ESCNN	Element spatial convolution neural network
IVP	Initial value problem
LST	Linear stability theory
LNSE	Linearized Navier-Stokes equation
MNLS	Modified nonlinear Schrödinger equation
MS	Multiple scales method
MSE	Mean square error
OSE	Orr-Sommerfeld equation
PDEs	Partial differential equations
PINN	Physics-informed neural networks
PSE	Parabolized stability equations
RNNs	Recurrent neural networks
SciANN	Scientific computational with artificial neural networks
SGD	Stochastic gradient descent
SNN	Standard neural network

LIST OF SYMBOLS

Latin symbols

a	Offset vector (bias) in neural networks
A, B, C, D	Parabolized stability equations matrix operators
c_i	Chebyshev spectral differentiation matrix coefficient
$D_{1,cheb}, D_{2,cheb}$	Chebyshev spectral differentiation matrices
E	Energy
$E(x)$	Disturbance kinetic energy
g_t	Gradient vector at timestep t
h_x, h_y	Curvature metrics
i	Imaginary number, $\sqrt{-1}$; imaginary part
L	Number of hidden layers in a neural network
m_t, v_t	First and second moment estimates in the ADAM optimizer
$\hat{m}_t, \hat{v}_t$	Bias-corrected moment estimates in the ADAM optimizer
N_y	Number of points in the wall-normal direction
p	Pressure
q	Flow state vector, $q = (u, v, w, p)$
$\bar{q}$	Steady base flow state vector
$\tilde{q}$	Unsteady disturbance state vector
$\hat{q}$	Disturbance amplitude state vector
r	Real part
Re	Reynolds number based on the flat plate length
Re_{δ_c}	Local Reynolds number based on the characteristic length scale
T	Temperature
u	Velocity component in the streamwise direction
U_e	Boundary layer edge velocity
U_∞	Freestream flow velocity
v	Velocity component in the wall-normal direction
w	Velocity component in the spanwise direction
W^k	Weight matrix between neural network layers
x	Spatial coordinate in the streamwise direction
y	Spatial coordinate in the wall-normal direction
$y_{cheb,i}$	Chebyshev nodes
$y_{phys,i}$	Coordinates in the physical domain
z	Spatial coordinate in the spanwise direction

Greek symbols

α	Streamwise wave number; Learning rate
$-\alpha_i$	Disturbance growth rate in the local, parallel stability problem
β	Spanwise wave number
β_1, β_2	Decay rates in the ADAM optimizer
δ	Boundary layer thickness
δ_c	Boundary layer characteristic length scale
ϵ	Constant for numerical stability in the ADAM optimizer
η	Learning rate
ζ	Term in the Parabolized Stability Equations (PSE) operators
Θ	Disturbance phase function
θ	Parameters of a stochastic objective function
λ	Parameter in a nonlinear differential operator
ν	Kinematic viscosity
ω	Angular frequency
Ω	Subspace in $\mathbb{R}^D$
ρ	Fluid density
σ	Disturbance growth rate in the spatial frame
σ_E	Disturbance kinetic energy-based growth rate
ψ	Stream function

CONTENTS

1 Introduction

The study of fluid dynamics plays a central role in engineering and applied sciences, as it encompasses problems of practical and theoretical interest such as aerodynamics, turbomachinery, and energy conversion. Among the many phenomena investigated within this field, the transition from laminar to turbulent flow represents one of the most challenging and relevant issues, directly impacting drag prediction, heat transfer rates, and overall system efficiency [1, 2]. Understanding and predicting such phenomena require advanced tools that can model the governing equations of fluid motion while maintaining computational feasibility.

Traditionally, the modeling of flows has been based on the solution of the Navier–Stokes equations, which are nonlinear partial differential equations (PDEs) governing continuum incompressible and compressible flows. While direct numerical simulation (DNS) provides a reference solution to these equations, the associated computational cost is prohibitive for most practical problems, especially at high Reynolds numbers [3]. Reduced-order models, linear stability theory (LST), and the Parabolized Stability Equations (PSE) have been introduced to mitigate this challenge, enabling the study of instabilities in boundary layers and the prediction of transition [4, 5]. Nonetheless, these approaches still involve significant computational expense and exhibit limitations, such as sensitivity to spurious modes and difficulties in handling three-dimensional or highly non-parallel flows [6].

1.1 Machine learning and PINNs in fluid dynamics

Over the past decade, the use of data-driven modeling combined with machine learning has gained increasing relevance in computational fluid dynamics (CFD). Techniques such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and autoencoders have been applied to flow prediction, turbulence modeling, and shape optimization [7, 8, 9]. Despite their success, purely data-driven models often require large training datasets and may lack physical consistency when extrapolating beyond the training regime.

To overcome these limitations, Physics-Informed Neural Networks (PINNs) have been introduced as a powerful alternative [10, 11, 12, 13, 14, 15]. PINNs integrate the governing equations of a physical system—expressed as PDEs—directly into the training process, reducing the dependency on large labeled datasets while ensuring that the model respects the underlying physics. As a result, PINNs are particularly well suited to fluid dynamics problems, where nonlinearities, high-dimensional spaces, and the scarcity of

reliable data poses challenges to standard machine learning methods. Its high capacity to handle problems with large numbers of variables makes it a viable solution method, as observed in [16] and [17]. Its ability to solve nonlinear partial differential equations is reported in the work of Yuan et al. [18], in which the technique is applied to the modified nonlinear Schrödinger equation (MNLS). Physics-informed neural networks can be applied to various mathematical models of fluid dynamics where conventional numerical methods face challenges [19]. Essentially, PINNs employ a given problem's underlying physics, known through governing equations written as partial differential equations (PDEs), to accelerate the training process. In general, an increased accuracy is also observed in the numerical results generated by the trained model when compared with standard neural network approaches. Smaller training data sets, when compared to standard neural networks, might be enough to accurately represent the desired physics. In CFD, the application of physics-informed neural networks primarily targets the Navier–Stokes equations. Therefore, PINNs are emerging as an alternative to the main numerical methods currently being used, such as CFD solvers based on the finite volume and the finite element methods [20]. Applications in aerodynamics and shape optimization are examples of the use of PINNs in applied fluid mechanics problems. Li et al. [8] demonstrate the applicability and prospects of machine learning applied to shape optimization, noting the advantages of some techniques and difficulties for large-scale optimization due to the high learning cost. Thus, PINNs present themselves as an alternative to overcoming this obstacle.

Different architectures of physics-informed neural networks are found in the literature. For instance, in [21] the authors proposed the Element Spatial Convolution Neural Network (ESCNN). The ESCNN technique exhibits superior learning performance compared to the state-of-the-art neural networks with fewer input parameters and better approximations in their results. The ESCNN approach uses the concept of convolutional neural networks. Through its application to an aerodynamic analysis problem, Gu et al. [9] demonstrate the ESCNN's ability to learn the physical laws governing the flow around an airfoil. Through its application, the learning of the Kutta condition, as well as the lift and drag coefficients prediction, is achieved. The numerical results were compared with those obtained through the panel method [22].

Another relevant extension of the PINN framework is the Variational Physics-Informed Neural Network (VPINN) [23]. This architecture is based on a Petrov–Galerkin formulation, in which the trial space is represented by neural networks and the test space by Legendre polynomials. By incorporating the variational form of the PDE into the loss function and integrating by parts, VPINNs reduce the order of differential operators that the neural network needs to approximate. As a result, this approach significantly decreases the training cost while improving accuracy compared to standard PINNs, as demonstrated in benchmark problems.

To address the spectral bias that is inherent in standard PINNs, the Multi-Scale Spatio-Temporal Fourier Features PINN (MS-ST-FF-PINN) has been proposed [24]. This architecture leverages Fourier feature embeddings to enhance the network's ability to capture both low- and high-frequency components of the solution. By doing so, MS-ST-FF-PINNs overcome the tendency of PINNs to converge only to low-frequency modes when dealing with problems characterized by multi-frequency spatial or temporal dynamics. Applications to canonical PDEs, such as the Poisson, wave, and Gray–Scott equations, as well as to the incompressible Navier—Stokes equations, demonstrate superior accuracy and faster convergence compared to standard PINNs, particularly in flows with periodic and turbulent features.

1.2 Problem statement and motivation

The accurate prediction of boundary layer development and the subsequent stability analysis are critical to understanding transition mechanisms. Classical approaches such as Blasius's similarity solution for flat plate boundary layers provide a theoretical foundation, while numerical methods based on finite differences and Runge–Kutta schemes enable accurate integration of base flows.

In the context of flow stability, LST and PSE methods have been extensively used to investigate the amplification of disturbances. Although PSE represents a significant advancement compared to LST, by accounting for non-parallel effects and allowing three-dimensional extensions, it still relies on complex numerical frameworks and is prone to numerical difficulties related to the computation of neutral points and the control of spurious modes [4, 6].

Given these challenges, there is strong motivation to investigate whether physics-informed neural networks can act as surrogate models for boundary layer and stability problems. If PINNs are capable of accurately reproducing base flow solutions and capturing disturbance evolution predicted by PSE, they may provide a computationally efficient, automated alternative to classical CFD solvers. Such models would be valuable not only for research but also for industrial applications in aerodynamics and flow control.

1.3 Objectives

The main goal of this dissertation is to explore the use of Physics-Informed Neural Networks in the modeling of laminar boundary layer flows and in flow stability analysis based on the parabolized stability equations. Specifically, this work implements a PINN framework for the solution of the flat-plate laminar boundary layer problem and compares its results with those obtained using a laminar boundary layer solver; assesses the performance

of PINNs in reproducing laminar boundary layer velocity profiles and the spatial evolution of the boundary layer relative to standard neural networks; investigates the application of PINNs to reconstruct pressure and velocity disturbances obtained from a nonlocal, nonparallel flow stability solver based on the parabolized stability equations; evaluates the ability of PINNs to reconstruct nonlocal, nonparallel stability-analysis eigenfunction fields in a sparse-data scenario; and employs the disturbance fields generated by the PINN to compute relevant metrics such as the energy-based disturbance growth rate (σ_E) and the N-factor.

1.4 Contributions

The main contributions of this work are as follows. First, it employs computational fluid dynamics solvers to generate training datasets for PINNs in boundary-layer and flow-stability problems, considering flows with low Reynolds numbers. Second, it evaluates the accuracy and efficiency of PINNs, compared to standard neural networks, in reconstructing base-flow solutions. Another important contribution is the investigation of PINNs for flow stability analysis using the PSE formulation, which has not yet been explored in the literature. Finally, this work provides insights into the potential of physics-informed machine learning techniques as alternatives to traditional CFD tools.

1.5 Dissertation overview

This dissertation is organized as follows. Chapter 2 presents the theoretical background of artificial neural networks, including the definitions of the artificial neuron, activation functions, weights, and optimizers. The concept of feedforward neural networks, which serve as the foundation for PINNs, is also introduced. Finally, the classical formulation of PINNs is presented, illustrated through an application to a problem governed by the Navier–Stokes equations. Chapter 3 presents a background on laminar boundary layer theory, including the Blasius similarity solution, as well as the numerical methods employed to generate the dataset used for training PINNs. Chapter 4 discusses the fundamentals of fluid flow stability, focusing on modal analysis, the Linear Stability Theory (LST), and the Parabolized Stability Equations (PSE). Chapter 5 introduces the Physics-Informed Neural Networks framework, describing the implementation adopted in this work and its application to the boundary layer and PSE problems. Chapter 6 presents the numerical results obtained, comparing PINNs with standard neural networks and reference solvers. Chapter 7 summarizes the conclusions of this research and outlines suggestions for future work.

2 Neural network theory

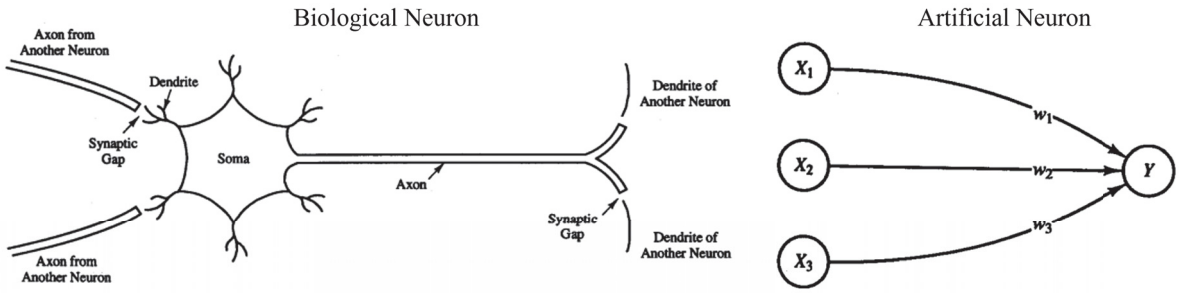
In this chapter, the basic concepts associated to neural networks in general are addressed, as well as a detailed explanation of the functioning of a feedforward neural network. We will also cover the theoretical formulation of the physics-informed neural network, which can be defined as a neural network that leverages the advantages of automatic differentiation to solve the governing equations of the physical problem being addressed.

2.1 Basics concepts of artificial neural network

In the field of machine learning, artificial neural networks are one of the most powerful tools for discovering patterns and information from data. Machine learning can be divided into three main groups: unsupervised learning, where information is extracted from unlabeled training data, semi-supervised learning, based on partially labeled training data or reinforcement learning, and finally, the group used in this work, supervised learning, in which the training data is labeled, that is, the network's outputs are then compared with the labeled data and are then optimized in order to decrease the error between them [7].

The first idea of an artificial neural network dates back to the 1940s, when McCulloch e Pitts [25] created the first mathematical representation of how a biological neuron functions. This concept is based on the idea that a biological neuron acts, in a simplified manner, as a computational element that can be described by propositional logic, operating in an "all-or-none" principle, a binary characteristic. Thus, artificial neural networks are generalizations of mathematical models that describe human cognition. The first example of an artificial neuron was developed by Rosenblatt [26], consisting of an input layer connected by paths to association neurons. The weights on these paths are adjusted iteratively until the problem under analysis is solved, meaning the network correctly reproduces the training inputs and outputs. In Figure 1, one can see the representation of a biological neuron and its counterpart, the artificial neuron. A neural network is characterized as a composition of several artificial neurons, also known as nodes, units, or cells. Each neuron is directly connected to other neurons through connection links, which have an associated weight, w. These weights are manipulated during training, being continuously adjusted in order to solve the problem. Each neuron in a network has an activation, also known as an activity level, which is a function of that neuron's inputs. This activation is then passed individually to other neurons through its connections, with signals being sent out one at a time [27].

Figure 1 – Representation of biological (left) and artificial (right) neurons.

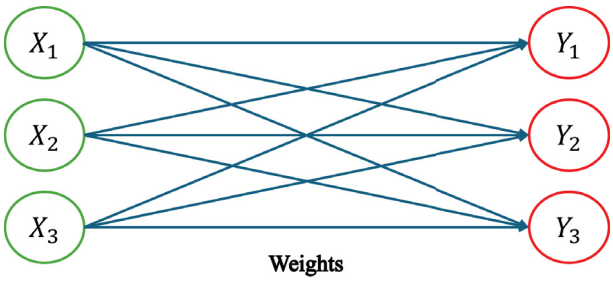


Source: Adapted from [27]

2.1.1 Single layer networks

The most basic form of a neural network is composed of just a single layer connected by weights to the output. The input layer is responsible for receiving external information, that is, data originating from images, tables, and other sources. Data is transmitted through weighted connections to the output layer, which represents the network's response to the input data, indicating whether or not the network has learned the problem at hand [28]. In Figure 2, one can observe the representation of a single-layer neural network. This type

Figure 2 – Single layer neural network representation.



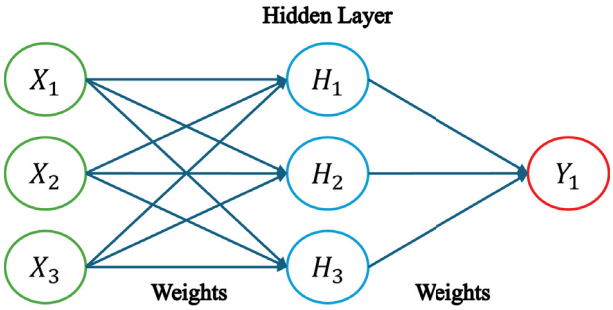
Source: The author (2025)

of neural network is simpler and requires less computational power when compared with multi-layer neural networks. Single layer networks are frequently used for problems such as simple image recognition, embedded systems applications, digital logic, and especially, educational applications.

2.1.2 Multiple layer networks

Neural networks with more than one layer represent greater computational power and encompass broader applications in complex problems. The intermediate layers located between the input and output layers lead to a more challenging training process. On the other hand, multiple layer networks make it possible to recognize more complex patterns and achieve higher success rates in problem-solving [28]. A representation of a neural network with an intermediate layer is shown in Figure 3.

Figure 3 – Representation of a neural network with an intermediate layer.



Source: The author (2025)

2.2 Activation function

The activation function is a crucial part of a neural network's learning process, enabling better training, learning, and predictive capabilities. It acts directly on each neuron along with the weights and bias, which will be discussed in more detail in subsequent sections. There are various activation functions, each with different implementations and ways of operating within the network. They exhibit different characteristics that make them suitable for specific types of problems, and they often must be tested to evaluate improvements in the network's generalization [29]. Some of the main types of activation functions used in the literature are presented below.

ReLU - Rectified Linear Units, is a type of activation function widely used in convolutional neural networks. It does not suffer from the vanishing gradient problem like the tanh and sigmoid activation functions. It can present the problem of dying neurons, i.e., neurons that become inactive throughout the entire training, instead of becoming inactive on specific occasions for the proper training of the network [30]. The ReLu activation function is expressed as:

$$F_{ReLU} = x^{+} = max(0, x). \tag{2.1}$$

Sigmoid - The Sigmoid activation function is mainly used for multi-class classification, mapping any value to an interval bounded by [0, 1] [31]. Its mathematical expression is given by:

$$F_{sigmoid} = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

SiLU - Sigmoid Linear Unit is a recently developed type of activation function that shows good performance in deep neural networks. It combines the simplicity of the ReLU activation function with the smoothness of the Sigmoid [32]. It is represented by the equation:

$$F_{silu} = x * \sigma(x). \tag{2.3}$$

Tanh - This activation function is a rescaled and centered version of the sigmoid function, mapping values to the interval [-1, 1] [33]. It is one of the most widely used activation functions in neural networks. Its formulation is as follows:

$$F_{Tanh} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.4}$$

2.3 Weight initialization

For proper training of the neural network, weight initialization is key. Indeed, during weight initialization the entire efficiency of the training is defined. Therefore, the choice of an appropriate weight initialization function must be made carefully, as the whole convergence and generalization capacity of the model depend on this initial step. For deep feedforward and recurrent neural networks, the most commonly recommended weight initialization schemes are Glorot/Xavier (normal or uniform) [34]. This method takes into account the number of input and output neurons of the network, ensuring that the variance of the activations in the forward pass and the variance of the gradients in the backward pass remain consistent across the layers. This initialization seeks to mitigate the problem of vanishing or exploding gradients during training, mainly caused when the weights are initialized randomly, which strongly harms the model's learning ability and generalization.

The technique can be addressed in two ways. In the first, called Glorot normal, the weights are defined through a normal distribution with mean 0 and standard deviation given by:

$$\sigma = \sqrt{\frac{2}{n_{in} + n_{out}}}. \tag{2.5}$$

The second is known as Glorot uniform, where the weights are defined from a uniform distribution in an interval $[-x, +x]$ given by:

$$x = \sqrt{\frac{6}{n_{in} + n_{out}}}. \tag{2.6}$$

In Eqns. (2.5) and (2.6), n_{in} and n_{out} represent the numbers of neurons in the input and output layers, respectively.

2.4 Optimizers in neural networks

Optimizers are an essential component in neural networks and are responsible for the weight update in the training process. They act in search of the best set of weights that minimize the loss function. The choice of the best optimizer depends on the neural network model being used [35]. Some optimizers are more utilized in neural networks, such as the Stochastic Gradient Descent (SGD) [36], which updates the model parameters in the

opposite direction of the gradient, calculating on a small sample of the data (mini-batch) at each iteration. It uses a single learning rate, which is constant and needs to be manually adjusted. The learning rate is a hyperparameter that controls the size of the steps taken by the optimizer during training, determining how quickly or slowly a neural network updates its parameters in response to the computed gradients. Another widely used optimizer is ADAGRAD, Adaptive Gradient [37], which is an optimizer that adapts the learning rate for each parameter individually. It does so by accumulating the squares of past gradients and decreasing the learning rate for parameters that receive frequent and large updates. The ADADELTA optimization method is an extension of ADAGRAD [38]designed to address the vanishing learning rate problem. Instead of accumulating all past gradients, it considers only a window of recent gradients. One of its main features is that it eliminates the need to define a global learning rate.

The optimizer used in this work is ADAM, which stands for Adaptive Moment Estimation, one of the most common models used in neural networks, developed by Kingma and Ba [39]. ADAM is an algorithm for first-order gradient-based optimization of stochastic objective functions. It is designed to be straightforward to implement, computationally efficient, and requires little memory. The method is particularly well-suited for problems that are large in terms of data or parameters and can handle non-stationary objectives and noisy and/or sparse gradients. The core idea of Adam is to combine the advantages of two other popular optimization methods: ADAGRAD, which works well with sparse gradients, and RMSProp, which performs well in online and non-stationary settings. The ADAM optimizer achieves this by computing individual adaptive learning rates for different parameters from estimates of the first and second moments of the gradients. The algorithm's primary function is to minimize the expected value of a stochastic objective function, $\mathbb{E}[f(\theta)]$, with respect to its parameters θ. At each timestep t, it receives a gradient vector $g_t = \nabla_\theta f_t(\theta)$, which is the vector of partial derivatives of the objective function at that step. Adam maintains two exponential moving averages, one for the gradient itself, the first moment, m_t, and one for the squared gradient, the second raw moment, v_t. These moving averages are controlled by the hyperparameter decay rates β_1 and β_2, respectively.

The first step of the algorithm is the update of the biased first moment estimate, in which the algorithm updates the moving average of the gradient. This acts like a momentum term, smoothing the updates and accelerating convergence in a consistent direction.

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t. \tag{2.7}$$

The second step is the update of the biased second raw moment estimate; it updates the moving average of the element-wise squared gradient. This term helps to adapt the learning rate for each parameter individually.

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \tag{2.8}$$

The third step is to compute bias-corrected moment estimates. Because the moment vectors m_0 and v_0 are initialized as vectors of zeros, the estimates in the first few steps are biased towards zero. Adam introduces a crucial step to correct for this initialization bias.

$$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}, \tag{2.9}$$

$$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}. \tag{2.10}$$

Finally, the parameters are updated using the bias-corrected moment estimates. The update is scaled by the square root of the second moment, which effectively creates a per-parameter learning rate.

$$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.11}$$

where α is the learning rate and ϵ is a small constant added for numerical stability. For a wide range of machine learning problems, the default, recommended values are: $\epsilon = 1 \times 10^{-8}$ [39].

The ADAM's main difference when compared against other algorithms like RMS-Prop is the bias correction step. The update rule for the second moment, v_t, can be expanded as a sum of all past squared gradients, weighted by the decay factor:

$$v_t = (1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} g_i^2. \tag{2.12}$$

To quantify the discrepancy between the estimated moment v_t and the true second moment, we analyze its expected value, $\mathbb{E}[v_t]$. Assuming a stationary true second moment, $\mathbb{E}[g_i^2] = \mathbb{E}[g_t^2]$ for all $i = 1, ..., t$, we can take the expectation of Eq. 2.12:

$$\mathbb{E}[v_t] = \mathbb{E}\left[(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} g_i^2\right] = \mathbb{E}[g_t^2](1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} = \mathbb{E}[g_t^2](1-\beta_2^t) \tag{2.13}$$

The result in Eq. 2.13 demonstrates that the expected value of the second moment estimate is the true moment scaled by a factor of $(1-\beta_2^t)$. This term is the source of the aforementioned initialization bias. To obtain a bias-corrected estimate, $\hat{v}_t$, we therefore divide the computed estimate, v_t, by this factor. The same correction is applied to the first moment, yielding the bias-corrected estimates used in the final parameter update rule, in Eqs. 2.9 and 2.10. This correction is of significant practical relevance. In scenarios involving sparse gradients, a high value for β_2 is necessary to reliably estimate the second moment by averaging over a larger number of gradients. It is precisely in this case that the lack of bias correction would lead to substantially larger initial steps, potentially causing algorithmic instability or divergence.

2.5 Feedfoward neural network

The general model of a feedforward neural network is composed of an input layer, one or more hidden layers, and an output layer. These components work together to learn the relationship between input and output data by adjusting their parameters (weights and biases). Each layer is composed of multiple neurons, which receive outputs from the previous layers (or the raw inputs, in the case of the first hidden layer) and calculate a weighted sum of these inputs, add an offset (bias), and then apply an activation function to produce its own output. Each hidden layer is formed by multiple neurons; each neuron receives one or more inputs with a certain weight and performs a mathematical operation, leading to an output [7]. In the context of model training, the sample can be defined as a pair **(x, y)** and the input vector is:

$$\mathbf{x} = (x_1, x_2, ..., x_m)^T, \tag{2.14}$$

where the subscript m is the input layer dimension and vector $\mathbf{y}$ represents the corresponding target outputs or labels that the neural network aims to predict. The output vector generated by the neural network is:

$$\mathbf{f} = (f_1, f_2, ..., f_c)^T, \tag{2.15}$$

where c is the neural network output layer dimension [40]. Considering that the number of hidden layers is $(k = 1, 2, ..., L)$, where L is the number of layers, with n_k neurons, one can demonstrate that the corresponding hidden layer vector is given by:

$$\mathbf{h} = (h_1, h_2, ..., h_k)^T.\mathbf{W}^k = (w_{ij}^k)_{n_k \times n_{k-1}}, \tag{2.16}$$

where $\mathbf{W}^k$ is the weight of matrices between the $(k-1)^{th}$ and k^{th} hidden layers. The weight matrix for the output layer can be represented by:

$$\mathbf{W}^{L+1} = (w_{ij}^{L+1})_{c \times n_L}. \tag{2.17}$$

The output of each layer can be represented by:

$$\begin{cases} \mathbf{h}_1 = \sigma_{h_1}(\mathbf{W}^1\mathbf{x} + \mathbf{a}^1), \\ \mathbf{h}_k = \sigma_{h_k}(\mathbf{W}^k\mathbf{h}_{k-1} + \mathbf{a}^k), \\ \mathbf{f} = \sigma_c(\mathbf{W}^{L+1}\mathbf{h}_R + \mathbf{a}^{L+1}), \end{cases} \tag{2.18}$$

where $\mathbf{a}$ is the offset, or bias, vector and σ is the activation function [41]. In this work, we use the Tanh activation function, given by Eq. 2.4.

In neural network design for regression, the definition of the loss function goes beyond an empirical choice. Alternatives such as Mean Absolute Error (MAE) or Huber Loss exist, frequently cited for their robustness with noisy data. However, this work uses

Mean Squared Error (MSE) as the primary metric. The use of MSE is not arbitrary, it represents the direct application of the Maximum Likelihood principle under the assumption that the target variable follows a normal (Gaussian) distribution [42]. This statistical property, coupled with the fact that MSE provides more informative gradients when the error is large, makes it the ideal choice for fitting continuous physical fields and ensuring the satisfaction of conservation laws:

$$E = \frac{1}{N}\sum_{i=1}^{N}|y - f|^2, \tag{2.19}$$

The weight $\mathbf{W}$ and the offset $\mathbf{a}$ are updated in order to minimize the errors in the prediction through the equations below:

$$\mathbf{W}^k = \mathbf{W}^k - \eta\frac{\partial E}{\partial \mathbf{W}^k}, \tag{2.20}$$

and

$$\mathbf{a}^k = \mathbf{a}^k - \eta\frac{\partial E}{\partial \mathbf{a}^k}, \tag{2.21}$$

where η is the learning rate [41].

2.6 Physics-informed neural network

The architecture and theory of physics-informed neural networks were developed by Raissi et al. [13], aiming to address nonlinear problems without the need for linearizations or adjustments at local temporal instances. These neural networks can be constrained according to the relevant physical laws, as dictated by the time-dependent, nonlinear partial differential equations governing the phenomenon of interest.

Initially, the problem of computing data-driven solutions for partial differential equations in its general form is defined,

$$\gamma_t + \mathcal{N}[\gamma;\lambda] = 0, \quad x \in \Omega, \quad t \in [0,T], \tag{2.22}$$

where $\gamma(t,x)$ is the latent solution, $\mathcal{N}[\cdot]$ is a nonlinear differential operator parametrized by λ, and Ω is a subspace of $\mathbb{R}^D$. By making $f(t,x)$ equal to the left side of equation (2.22), one gets,

$$f = \gamma_t + \mathcal{N}[\gamma;\lambda]. \tag{2.23}$$

Through the definition in Eq. (2.23), we have a physics-informed neural network, approximating $f(t,x)$ by a deep neural network. One can apply the chain rule to this neural network for a composition of differential functions through automatic differentiation [43], as described in Ref. [44].

To illustrate this approach for a fluid mechanics problem, the author presents an application of physics-informed neural networks to a problem involving the Navier-Stokes

equations, which describe a range of physical problems of engineering interest. Considering the two-dimensional incompressible Navier-Stokes equations, we have:

$$u_t + \lambda_1(uu_x + vu_x) = -p_x + \lambda_2(u_{xx} + u_{yy}), \tag{2.24}$$

$$u_t + \lambda_1(uv_x + vv_x) = -p_y + \lambda_2(v_{xx} + v_{yy}), \tag{2.25}$$

where u and v denote the velocity components aligned with the x and y axes, respectively, p is the pressure, and the subscripts represent derivative terms. For this case, $\lambda = (\lambda_1, \lambda_2)$ are unknown parameters.

The last equation to complete the system is the conservation of mass for incompressible flow, represented by:

$$u_x + v_y = 0. \tag{2.26}$$

Assuming that a stream function exists such that:

$$u = \psi_y, \quad v = -\psi_x, \tag{2.27}$$

the continuity equation, 2.26, will be automatically satisfied. The unknown parameters, λ, and pressure, p, are the variables learned by the neural network. We then define $f(t, x, y)$ and $g(t, x, y)$ as:

$$f = u_t + \lambda_1(uu_x + vu_x) + p_x - \lambda_2(u_{xx} + u_{yy}), \tag{2.28}$$

$$g = u_t + \lambda_1(uv_x + vv_x) + p_y - \lambda_2(v_{xx} + v_{yy}). \tag{2.29}$$

The neural network is then trained to learn these two output parameters, $\psi(t, x, y)$ and $p(t, x, y)$. Based on the definitions given in Eqs. 2.27, 2.28, and 2.29, the physics-informed neural network is defined. In this formulation, f and g act on the neural network and are minimized through the loss function computed using the mean square error:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(|u(t^i, x^i, y^i) - u^i|^2 + |v(t^i, x^i, y^i) - v^i|^2) + \frac{1}{N}\sum_{i=1}^{N}(|f(t^i, x^i, y^i)|^2 + |g(t^i, x^i, y^i)|^2). \tag{2.30}$$

This approach is then applied to flow around a cylinder, a typical problem in fluid mechanics.

2.7 Machine learning frameworks used in this work

The implementation of the neural network models in this work was developed using high-level deep learning libraries in Python. The choice of the computational framework is a fundamental step, as it dictates the flexibility and implementation capability of the proposed models. Initially, the chosen framework for development was SciANN [45], a Python package specifically designed for scientific computing and physics-informed deep learning (PINN). SciANN acts as a high-level wrapper for the Keras and TensorFlow libraries, aiming to

abstract the construction of neural networks for the solution and discovery of partial differential equations (PDEs). This tool inherits important functionalities from its backends, such as batch optimization, model reuse for transfer learning, and, crucially, graph-based automatic differentiation. Although SciANN provides a powerful abstraction for PINN architectures, limitations were found in adapting its code to the specific requirements and complexity of the present study. To achieve greater flexibility and finer control over the model architecture and training cycle, it was decided to implement the models directly using the underlying libraries. Thus, the final implementation was developed with TensorFlow 2 [46] and its official high-level API, Keras [47]. TensorFlow is an open-source platform for large-scale machine learning, offering a comprehensive ecosystem of tools and libraries. Its main strength lies in high-performance numerical computation, with support for CPU/GPU parallelization, and its robust automatic differentiation engine, essential for gradient-based optimization. Keras, in turn, serves as the high-level interface of TensorFlow, designed to allow rapid prototyping and intuitive, modular model construction. The combination of these two tools provided the ideal balance between low-level control and the flexibility of TensorFlow with the ease of use and development speed of Keras, being the definitive choice for the implementation in this work.

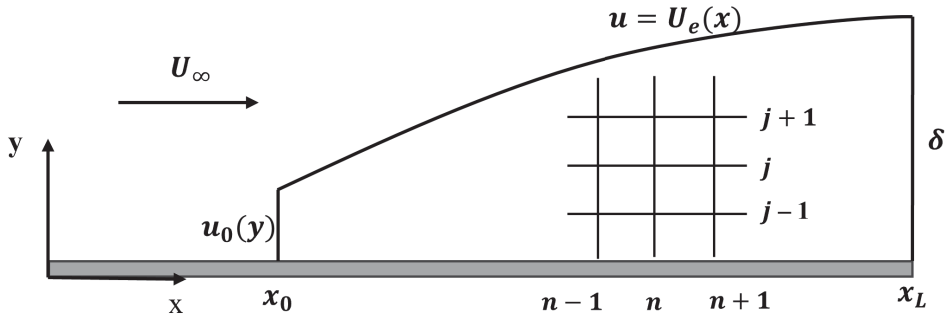
3 The laminar boundary layer

In this chapter, the theoretical background of the laminar boundary layer over a flat plate is presented, as well as the numerical solution developed for the generation of the dataset used to train the neural network. This chapter includes, therefore, the laminar boundary layer equations for incompressible two-dimensional flow and the finite difference-based discretization employed for the numerical solution of the boundary layer equations. This chapter also details the boundary layer initialization through Blasius velocity profiles.

3.1 Laminar boundary layer equations

Boundary layers form in regions adjacent to surface walls as a result of viscous effects. In Fig. 4, one observes the development of a boundary layer over a flat plate. The freestream velocity is U_∞. The boundary layer thickness, represented by δ in Fig. 4, is the most commonly used quantity to describe the boundary layer extent. The boundary layer thickness is defined as the distance $y = \delta$ from the surface where the local velocity, $u(y)$, reaches 99% of the free-stream velocity [48].

Figure 4 – Boundary layer development over a flat plate.



Source: The author (2025).

This section is devoted to deriving the governing equations of the flow within the boundary layer starting from the Navier-–Stokes equations. The derivation may proceed along two paths: a physically based argument, originally employed by Prandtl, or a mathematical limiting procedure. The present text follows the physical approach, as it provides clearer intuition regarding the simplifications introduced [48].

The analysis is based on a boundary layer configuration over a flat surface or one with gentle curvature, where the boundary layer thickness, δ, is very small compared to

the radius of curvature. In such problems, the main geometric length scale is the distance x from the leading edge of the surface. A fundamental premise is that, except very close to the leading edge, the boundary layer thickness is much smaller than the distance x, i.e., $\delta/x \ll 1$.

Prandtl's physical derivation is based on an order-of-magnitude analysis of the terms in the Navier–Stokes equations. Within the boundary layer, the following orders of magnitude are established:

- The velocity component in the x-direction, u, is of the order of the external flow velocity, U:
$$u \sim U$$
- The variation in the x-direction, $\partial/\partial x$, is of the order of $1/x$:
$$\frac{\partial}{\partial x} \sim \frac{1}{x}$$
- From the continuity equation, $\partial v/\partial y$ must be of the same order as $\partial u/\partial x$, i.e., U/x.
- Since the variation in the y-direction, $\partial/\partial y$, is much larger than in the x-direction (of the order of $1/\delta$), the velocity component v must be of the order of $U\delta/x$:
$$v \sim U\frac{\delta}{x} \quad \text{and} \quad \frac{\partial}{\partial y} \sim \frac{1}{\delta}$$

The Navier-–Stokes equations for a two-dimensional steady flow are given by:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\frac{\partial^2 u}{\partial x^2} + \nu\frac{\partial^2 u}{\partial y^2}. \tag{3.1}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\frac{\partial^2 v}{\partial x^2} + \nu\frac{\partial^2 v}{\partial y^2}. \tag{3.2}$$

Applying the order-of-magnitude analysis to each term of the equation in the x-direction, we obtain:

$$\frac{U^2}{x} + \frac{U^2}{x} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\frac{U}{x^2} + \nu\frac{U}{\delta^2}.$$

Based on this balance, the second viscous term ($\nu\partial^2 u/\partial y^2$) is much larger than the first ($\nu\partial^2 u/\partial x^2$), allowing the latter to be neglected. It is assumed that, due to the presence of strong viscous effects and particle acceleration, the dominant viscous term must have the same order of magnitude as the inertia terms. This leads to the important relation:

$$\frac{U^2}{x} \sim \nu\frac{U}{\delta^2} \quad \Rightarrow \quad \delta \sim \sqrt{\frac{\nu x}{U}}.$$

This reasoning, purely based on order-of-magnitude analysis, indicates that the boundary layer thickness grows with $\sqrt{x}$. Furthermore, the initial premise that $\delta/x \ll 1$ is equivalent to the condition that the Reynolds number, $Re = Ux/\nu$, is large ($Re \gg 1$).

With these considerations in mind, the x-component of the Navier-–Stokes equations is approximated, for a boundary layer, by:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\frac{\partial^2 u}{\partial y^2}. \tag{3.3}$$

For the equation in the y-direction, the analysis shows that both inertia and viscous terms are of an order of magnitude δ/x smaller than their counterparts in the x-direction, and can, therefore, be neglected. The equation then reduces to:

$$0 = -\frac{1}{\rho}\frac{\partial p}{\partial y}. \tag{3.4}$$

This implies that the pressure, p, is independent of the wall-normal coordinate y inside the boundary layer, being only a function of x, that is, $p = p(x)$. Therefore, the incompressible boundary layer governing equations are:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0. \tag{3.5}$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{dp}{dx} + \nu\frac{\partial^2 u}{\partial y^2}. \tag{3.6}$$

$$\frac{\partial p}{\partial y} = 0. \tag{3.7}$$

Since the pressure p depends only on x, and the pressure gradient in the normal direction, $\frac{\partial p}{\partial y}$, vanishes within the boundary layer, its distribution along the boundary layer is identical to that of the external inviscid flow. For incompressible flows, such as those considered in this work, the region outside the boundary-layer edge behaves as an irrotational (potential) flow. Therefore, in this region, Bernoulli's equation is valid [49]:

$$\frac{p}{\rho} + \frac{1}{2}U^2 = \text{constant}. \tag{3.8}$$

Differentiating this equation with respect to x, we obtain an expression for the pressure gradient:

$$-\frac{1}{\rho}\frac{dp}{dx} = U_e\frac{dU_e}{dx}, \tag{3.9}$$

where U_e is the velocity at the boundary layer edge. By substituting this result into Eq. (3.6), we arrive at an alternative form of Prandtl's boundary layer equation:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = U_e\frac{dU_e}{dx} + \nu\frac{\partial^2 u}{\partial y^2}. \tag{3.10}$$

The boundary layer equations must be solved with a set of boundary conditions that ensure the no-slip condition at the surface and the matching with the external flow velocity far from the surface.

The set of equations obtained above is known as the incompressible boundary layer equations, which capture the essential physics of such flows [2, 50],

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{3.11}$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = U_e\frac{\partial U_e}{\partial x} + \nu\frac{\partial^2 u}{\partial y^2}, \tag{3.12}$$

$$\frac{\partial p}{\partial y} = 0. \tag{3.13}$$

The boundary layer momentum equation in the streamwise direction is parabolic in nature. Therefore, its numerical solution can be performed through a spatial marching scheme. This aspect will be further discussed in Sec. 3.2. The boundary conditions for Eqns. (3.11), (3.12), and (3.13) are:

No slip at the wall:	$y = 0,\ u(x,0) = v(x,0) = 0,$
Far from the wall:	$y = \infty,\ u(x,\infty) = U_\infty,\ v(x,\infty) = 0$
Inflow conditions:	$x = 0,\ u(0,y) = u_0(0,y),\ v(0,y) = 0$

where U_∞ is the freestream velocity and u_0 is the inflow velocity profile. The outer boundary condition, $u(x,\infty) = U_\infty$, couples the internal boundary-layer solution to the external flow. Therefore, the external potential-flow solution must be known before the boundary-layer equations can be solved..

3.2 Boundary layer numerical solution

The numerical solution of the boundary layer is obtained in the computational domain illustrated in Fig. 4. As the inlet condition at x_0, the velocity profile from the Blasius similarity solution is adopted. This profile serves as the starting point for the numerical method, which advances the solution using a second-order finite difference scheme.

3.2.1 Similarity velocity profiles

An exact solution to the boundary layer equations, shown in Eqs. (3.11) and (3.12), was obtained by Blasius [51]. He proposed that the problem admits a self-similar solution. This means that the velocity profiles at different positions x along the plate are geometrically similar and can be simplified into a single universal curve if the variables are properly scaled. To reduce Eqs. (3.11) and (3.12) to a single ordinary differential equation, the stream function is defined as:

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}. \tag{3.14}$$

This satisfies the continuity equation for all stream functions ψ and yields the following boundary layer equation [48]:

$$\frac{\partial \psi}{\partial y}\frac{\partial^2 \psi}{\partial x \partial y} - \frac{\partial \psi}{\partial x}\frac{\partial^2 \psi}{\partial y^2} = \nu\frac{\partial^3 \psi}{\partial y^3}. \tag{3.15}$$

The similarity solution for this problem then assumes the form:

$$\psi(x, y) \approx f(\eta), \tag{3.16}$$

where η is given by:

$$\eta = \frac{y}{\sqrt{\nu x / U_\infty}}, \tag{3.17}$$

the coefficients ν and U_∞ make the variable η dimensionless. Physically, since the boundary layer thickness grows proportionally to $\sqrt{x}$, the stream function must also incorporate this term to ensure that the velocity profiles are self-similar, while also guaranteeing that ψ is, indeed, dimensionless. Thus, the stream function assumes the form:

$$\psi(x, y) = \sqrt{\nu U_\infty x} f(\eta). \tag{3.18}$$

Substituting this definition into the derivative terms that compose the boundary layer equation, Eq. (3.15), we obtain:

$$\frac{\partial \psi}{\partial x} = -\frac{1}{2}\sqrt{\frac{\nu U_\infty}{x}} \eta f' + \frac{1}{2}\sqrt{\frac{\nu U_\infty}{x}} f, \tag{3.19}$$

$$\frac{\partial \psi}{\partial y} = U f', \tag{3.20}$$

$$\frac{\partial^2 \psi}{\partial x \partial y} = -\frac{U_\infty}{2x} \eta f'', \tag{3.21}$$

$$\frac{\partial^2 \psi}{\partial y^2} = U_\infty \sqrt{\frac{U_\infty}{\nu x}} f'', \tag{3.22}$$

$$\frac{\partial^3 \psi}{\partial y^3} = \frac{U_\infty^2}{\nu x} f'''. \tag{3.23}$$

The resulting ordinary differential equation, after substituting the derivative terms, is:

$$f''' + \frac{1}{2} f f'' = 0, \tag{3.24}$$

which must satisfy the following boundary conditions [49]:

$$f(0) = f'(0) = 0, \tag{3.25}$$

$$f'(\eta) \to 1 \quad as \quad \eta \to \infty. \tag{3.26}$$

The methodology used to solve the Blasius equation is a combination of three numerical techniques that constitute the shooting method. First, since the Blasius equation is a Boundary Value Problem (BVP) with an unknown boundary condition, ($f''(0)$), the strategy is to transform the problem into an equivalent Initial Value Problem (IVP). This is done by estimating a value for $f''(0)$. With a complete set of initial conditions, the Blasius equation is then solved numerically using a robust integrator, which in this case is the 4th-order Runge—Kutta method. After the integration, the computed value at the

other boundary, $f'(\eta_{max})$, is compared with the target value, and the difference constitutes an error. To intelligently correct the initial estimate and converge quickly to the solution, a root-finding algorithm is employed, specifically, the secant method. This method uses the results of previous shots to systematically refine the estimate of $f''(0)$, repeating the process until the error is minimized and the boundary condition is satisfied with the desired accuracy.

The pseudocode in Algorithm 1 summarizes the shooting method numerical procedure as applied to the Blasius boundary layer problem. The algorithm starts by guessing an initial value for the unknown boundary condition, $f''(0)$, and solving the resulting initial value problem with the Runge–Kutta method. The discrepancy between the computed solution at the farfield boundary and the expected asymptotic condition defines the residual error. To update the guess efficiently, the secant method is employed, combining information from successive iterations to approximate the root of the residual function. This iterative loop continues until the error falls below a prescribed tolerance, ensuring convergence to the correct similarity solution.

Algorithm 1: Shooting Method for Boundary Value Problems

▷ `Input: Boundary conditions` $y(x_0) = y_0$, $y(x_f) = y_f$; `initial guesses` s_0, s_1

$s_{\text{old}} \leftarrow s_0$
$s \leftarrow s_1$
▷ `Solve IVP for initial guesses to start the secant method`

Solve $f''' + \frac{1}{2} f f'' = 0$ with $y(x_0) = y_0, y'(x_0) = s_{\text{old}}$
$R_{\text{old}} \leftarrow y(x_f; s_{\text{old}}) - y_f$
Solve $f''' + \frac{1}{2} f f'' = 0$ with $y(x_0) = y_0, y'(x_0) = s$
$R \leftarrow y(x_f; s) - y_f$
while $|R| > tolerance$ **do**
  $s_{\text{new}} \leftarrow s - R\frac{s - s_{\text{old}}}{R - R_{\text{old}}}$ ▷ `Update slope with the secant method`

  $s_{\text{old}} \leftarrow s$
  $s \leftarrow s_{\text{new}}$
  $R_{\text{old}} \leftarrow R$
  ▷ `Solve the BVP with the new slope guess`

  Solve $f''' + \frac{1}{2} f f'' = 0$ with $y(x_0) = y_0, y'(x_0) = s$
  $R \leftarrow y(x_f; s) - y_f$
▷ `Output: The converged solution` $y(x)$ `for the final slope` s

3.2.2 Finite differences aproximation

To develop a numerical solution for the laminar boundary layer, we employ a second-order finite difference scheme on a uniform grid for the derivative terms, as described in Ref. [50]. The velocity derivatives are computed as follows:

$$\begin{aligned}
\frac{\partial u}{\partial x} &\approx \frac{1.5u_j^{n+1} - 2u_j^n + 0.5u_j^{n-1}}{\Delta x} + O(\Delta x^2), \\
\frac{\partial u}{\partial y} &\approx \frac{u_{j+1}^{n+1} - u_{j-1}^{n+1}}{2\Delta y} + O(\Delta y^2), \\
\frac{\partial^2 u}{\partial y^2} &\approx \frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta y^2} + O(\Delta y^2),
\end{aligned} \tag{3.27}$$

where j represents the wall-normal index and n represents the flow direction index. Notice that, because the boundary layer problem is parabolic, it is solved through a marching numerical scheme, hence the choice of n for the streamwise index counter, following Ref. [50].

In Eq. (3.12), the velocity components that appear undifferentiated, u and v, are extrapolated in the streamwise (marching) direction as follows:

$$\begin{aligned}
u_j^{n+1} &\approx 2u_j^n - u_j^{n-1} + O(\Delta x^2), \\
v_j^{n+1} &\approx 2v_j^n - v_j^{n-1} + O(\Delta x^2).
\end{aligned} \tag{3.28}$$

where the superscript n denotes the grid line in the streamwise direction. By replacing the undifferentiated velocities at $n+1$ with the extrapolated values in (3.28), the momentum equation becomes linear in the unknowns u^{n+1}, and a linear system for the vector $\mathbf{u}^{n+1}$ (assembled over all j) can be formed and solved. Note that this explicit extrapolation is second-order accurate in Δx and may affect numerical stability, appropriate stabilization or step-size control may therefore be required.

The previous expressions are then substituted back into Eqs. (3.12), and subsequently rearranged to generate a tridiagonal system of equations associated with the grid line $n+1$ through the boundary layer:

$$\begin{aligned}
&a_j u_{j-1}^{n+1} + b_j u_j^{n+1} + c_j u_{j+1}^{n+1} = d_j, \\
&a_j = -\frac{\Delta x}{2\Delta y}(2v_j^n - v_j^{n-1}) - \nu\frac{\Delta x}{\Delta y^2}, \\
&b_j = 1.5(2u_j^n - u_j^{n-1}) - 2\nu\frac{\Delta x}{\Delta y^2}, \\
&c_j = \frac{\Delta x}{2\Delta y}(2v_j^n - v_j^{n-1}) - \nu\frac{\Delta x}{\Delta y^2}, \\
&d_j = (2u_j^n - u_j^{n-1})(2u_j^n - 0.5u_j^{n-1}) + \Delta x\left(U_e\frac{dU_e}{dx}\right)^{n+1},
\end{aligned} \tag{3.29}$$

Since the boundary layer code is formulated for a zero-pressure-gradient flow, the last term of the in d_j in Eq. (3.29) is neglected. From Eq. (3.11) it is possible to obtain v_j^{n+1},

since u_j^{n+1} is calculated from the solution of the tridiagonal system in Eq. (3.29):

$$v_j^{n+1} = v_{j-1}^{n+1} - 0.5\frac{\Delta y}{\Delta x}[(1.5u_j^{n+1} - 2u_j^n + 0.5u_j^{n-1}) + (1.5u_{j-1}^{n+1} - 2u_{j-1}^n + 0.5u_{j-1}^{n-1})], \quad (3.30)$$

where for the first element of the grid in normal direction, $j = 1$, $v_1^{n+1} = 0$. Although the velocity components u and v are explicitly extrapolated to construct the coefficients of the discretized equations, the scheme remains implicit because the unknown u^{n+1} is obtained from the solution of a tridiagonal system that couples the values u_{j-1}^{n+1}, u_j^{n+1}, and u_{j+1}^{n+1}. Therefore, the formulation is implicit in the wall-normal direction. The resulting linear system is solved using a tridiagonal matrix algorithm [52]. The length of the domain in the direction normal to the flow, y, is defined through the following expression [2]:

$$\delta_{99} = 5\sqrt{\frac{\nu x}{U_\infty}}, \quad (3.31)$$

making y_{max} equal to $1.5\delta_{99}$.

The pseudocode in Algorithm 2 summarizes the numerical methodology used to approximate the two-dimensional boundary layer development over a flat plate. The algorithm begins by defining the Reynolds number, cinematic viscosity, and discretized computational domain in the streamwise and wall-normal directions. At the inlet, the velocity profiles are initialized based on the Blasius solution to provide a realistic starting condition. The governing equations are then discretized and solved iteratively. The u-velocity component is obtained from the discretized momentum equation using a tridiagonal matrix system solved by forward elimination and back substitution, while the v-velocity component is computed from the continuity equation in its discrete form. This marching procedure in the x-direction allows the algorithm to progressively build the velocity field along the plate. The methodology combines analytical insights from the similarity solution with numerical stability provided by implicit finite-difference schemes, ensuring an accurate representation of the boundary layer growth.

Algorithm 2: Boundary Layer Computation

▷ Input: Flow parameters U_∞, ν; computational domain $x_{\texttt{mesh}}, y_{\texttt{mesh}}$

$\mathbf{u}, \mathbf{v} \leftarrow \mathbf{0}$ ▷ Initialize velocity fields

▷ Set inlet profiles at n=0 and n=1 using Blasius solution

$\mathbf{u}^0, \mathbf{v}^0 \leftarrow$ Blasius Solution$(y_{\text{mesh}}, U_\infty, \nu)$
$\mathbf{u}^1, \mathbf{v}^1 \leftarrow$ Blasius Solution$(y_{\text{mesh}}, U_\infty, \nu)$
for *each streamwise station n from* 2 *to* $n_{max} - 1$ **do**
  ▷ Solve the discretized momentum equation for $\mathbf{u}^{n+1}$

  $a_j u_{j-1}^{n+1} + b_j u_j^{n+1} + c_j u_{j+1}^{n+1} = d_j$ ▷ for $j = 1, \ldots, j_{\texttt{max}}$

  ▷ Update $\mathbf{v}^{n+1}$ from the discretized continuity equation

  $v_j^{n+1} \leftarrow v_{j-1}^{n+1} - 0.5\frac{\Delta y}{\Delta x}[(1.5u_j^{n+1} - 2u_j^n + 0.5u_j^{n-1}) + \ldots]$
▷ Output: Discretized velocity fields $\mathbf{u}$ and $\mathbf{v}$ over the domain

4 Linear stability analysis

The study of flow stability is relevant, among others, in the analysis of the transition from laminar to turbulent regimes, considered one of the central problems in fluid mechanics [1]. The analysis of instabilities in flows is generally carried out using the modal approach [5], with the classical theory being the Linear Stability Theory (LST), based on the Orr–Sommerfeld equation (OSE) [2]. LST has been widely used to predict transition [53], but it presents limitations since it does not directly account for boundary layer growth, non-parallel effects, or curvature. Practical transition prediction methods, such as the e^N method, were introduced by Smith and Gamberoni [54] and Van Ingen [55], showing good agreement with experiments [56, 57].

To overcome the limitations of Linear Stability Theory, nonlocal and nonparallel stability analyses were developed, such as solutions based on the Wentzel–Kramers–Brillouin -Jeffreys approximation, WKJB [58], and the multiple scales method (MS) [59, 60], which incorporates nonparallel effects in the boundary layer arising from the fact that the wall-normal velocity component is nonzero. Another important advancement in stability analysis are the Parabolized Stability Equations (PSE) [61, 4], a nonlocal and nonparallel method capable of modeling the evolution of convectively unstable disturbances in boundary layers, including curvature effects. The classical PSE formulation is of the 2.5D type, which means that the base flow exhibits slow variations in the spanwise direction.

For cases in which the variation in the spanwise direction is considerably larger, the fully 3D PSE formulation should be employed [62]. The PSE approach has a computational cost comparable to that of LST, but with results close to DNS, although at a cost two orders of magnitude lower [5]. By using PSE, growth rates are obtained from the superimposed disturbances for a given base flow; this growth rate then results, through integration, in the N-factor, which is used to predict the transition region [55] in the e^N method. The widespread use of PSE, particularly in industry, still presents some challenges. For instance, the correct computation of the N-envelope using PSE depends on the knowledge of the neutral point location for each disturbance mode considered in the integration. Another challenge is that, depending on the strategy adopted for mode generation, spurious disturbance modes can lead to unbounded amplification in PSE calculations [6].

4.1 Modal linear stability analysis

In modal linear stability theory, the stability problem is solved using a set of wave modes, where each mode is analyzed independently. Both Linear Stability Theory (LST) and the Parabolized Stability Equations (PSE) are examples of modal linear stability

methods.

The fundamental principle is based on the decomposition of any flow property, q, into a steady base state, $\overline{q}$, and an unsteady perturbation component, $\tilde{q}$. This decomposition is expressed as [5]:

$$q(x,t) = \overline{q}(x) + \epsilon\tilde{q}(x,t), \tag{4.1}$$

where x is the spatial coordinate vector, t is time, and q is the flow state vector containing properties such as velocity components, and pressure, $q = (u, v, w, p)$.

In the context of linear stability, the perturbations have small amplitudes, so that $\epsilon \ll 1$. By applying this approach to the Navier–Stokes equations, a linearization is performed by neglecting terms of order $\mathcal{O}(\epsilon^2)$ or higher. Next, the equations for the steady flow are subtracted, resulting in the Linearized Navier–Stokes Equations (LNSE). The linearization is valid only while the perturbation amplitude remains small, such that the nonlinear terms are negligible. It is important to note that a real nonlinear system can exhibit unstable behavior for finite-amplitude perturbations, even under conditions where the linearized system remains stable.

If the base flow $\overline{q}$ is steady, the time and spatial dependencies can be separated. A Fourier decomposition in time can be introduced using the form $\tilde{q} = \hat{q}e^{-i\omega t}$, where ω is the angular frequency. In general, in modal stability analysis, the perturbation term is written as the product of an amplitude $\hat{q}$ and a phase function Θ:

$$\tilde{q} = \hat{q}e^{i\Theta}. \tag{4.2}$$

Different stability methods, such as LST and PSE, rely on distinct assumptions regarding the topology of the base flow $\overline{q}$ and the fluctuation $\tilde{q}$. The main differences between the LST and PSE approaches are detailed in Tab. 1, where $\alpha = 2\pi/L_x$ is the streamwise wave number, $\beta = 2\pi/L_z$ is the spanwise wave number, with L_x and L_z the wavelengths in the streamwise (x) and spanwise (z) directions, respectively. Also, ω is the angular frequency, and x^* represents the base flow slow variation properties in the streamwise direction. The wall-normal direction is denoted by y.

Table 1 – Classification of stability analysis theory [5].

Method	Assumption	Base flow	Amplitude	Phase Θ
PSE	$\partial_x\mathbf{q} \ll \partial_y\mathbf{q}$; $\partial_z\mathbf{q} = 0$	$\bar{\mathbf{q}}(x^*, y)$	$\hat{\mathbf{q}}(x^*, y)$	$\int \alpha(x')dx' + (\beta z - \omega t)$
LST	$\partial_x\mathbf{q} = \partial_z\mathbf{q} = 0$	$\bar{\mathbf{q}}(y)$	$\hat{\mathbf{q}}(y)$	$\alpha x + (\beta z - \omega t)$

4.1.1 Linear stability theory

The Linear Stability Theory (LST) approach represents the solution of the Orr–Sommerfeld equation and considers the assumption of a local and parallel base flow. The main assump-

tions of LST are:

- **Base flow:** The flow is considered parallel, i.e., its derivatives in the streamwise (x) and spanwise (z) directions are zero ($\partial_x \overline{q} = \partial_z \overline{q} = 0$), depending only on the wall-normal coordinate, $\overline{q}(y)$.
- **Amplitude:** The perturbation amplitude also varies only in the wall-normal direction, $\hat{q}(y)$.
- **Phase:** The phase function is linear in the spatial coordinates, $\Theta = \alpha x + \beta z - \omega t$.

Due to the assumption of local flow, non-local effects, such as derivatives of the base flow in the streamwise direction, are not considered, and the stability problem becomes an eigenvalue problem. Curvature effects are also not included [63].

The LST equation for compressible flows can be expressed in a compact form as:

$$\frac{d^2\hat{\mathbf{q}}}{dy^2} + A_L \frac{d\hat{\mathbf{q}}}{dy} + B_L \hat{\mathbf{q}} = \omega C_L \hat{q} \tag{4.3}$$

where $\mathbf{A}_L$, $\mathbf{B}_L$, and $\mathbf{C}_L$ are operators related to the base flow parameters and the wavenumbers α and β.

To solve the problem, it is advantageous to consider the temporal formulation rather than the spatial one. In the spatial problem, the streamwise wavenumber, α, is complex ($\alpha = \alpha_r + i\alpha_i$), while the angular frequency, ω, is real [60]. In the temporal problem, α is real and ω is complex ($\omega = \omega_r + i\omega_i$). Gaster transformation [64] is then used to convert temporal-domain results to the spatial domain, which is required to initialize the PSE method. In the spatial frame, the wave growth rate, σ, is defined as $\sigma = -\alpha_i$. The neutral point corresponds to the location where the stability mode has zero growth rate, i.e., $\alpha_i = 0$. The neutral point location can be determined by starting a calculation within the unstable region (where $\alpha_i < 0$) and marching upstream until the condition $\alpha_i = 0$ is satisfied.

The Gaster transformation associates wave numbers as obtained in the temporal (subscript T) and spatial (subscript S) theories as follows

$$\alpha_{r,S} \approx \alpha_{r,T}, \tag{4.4}$$

$$\omega_{r,S} \approx \omega_{r,T}, \tag{4.5}$$

$$\frac{\omega_{i,T}}{\alpha_{i,S}} \approx -\frac{\partial \omega_r}{\partial \alpha_r}, \tag{4.6}$$

where the subscripts r and i denote the real and imaginary parts, respectively. In the spatial framework, the parallel, local wave mode growth rate, σ, is equivalent to the imaginary part of the streamwise wavenumber with a switch in sign: $\sigma = -\alpha_i$. The neutral

point corresponds to the position where the stability mode has zero growth rate. It is possible to determine the neutral point location by starting a computation inside the unstable region, where $\alpha_i < 0$, and move upstream until $\alpha_i = 0$, defining the neutral point.

4.1.2 PSE theory

The PSE method is appropriate for the analysis of streamwise disturbance growth in slowly varying shear flows such as boundary layers [65, 66, 4]. Mathematically, this is expressed as

$$\partial_x \bar{\mathbf{q}} \ll \partial_y \bar{\mathbf{q}}; \quad \partial_z \bar{\mathbf{q}} = \mathbf{0}, \tag{4.7}$$

$$\bar{\mathbf{q}}(\mathbf{x}) = \bar{\mathbf{q}}(x^*, y), \tag{4.8}$$

where x^* is a scaled version of x used to represent the base flow slow variation in the x direction.

The base flow velocity components $\bar{u}$ and $\bar{v}$, aligned with the streamwise and wall-normal directions, respectively, exhibit small variations in the streamwise (x) direction and are constant along the spanwise (z) direction. We introduce the local Reynolds number, $Re_{\delta_c} = U_e \delta_c(x)/\nu$ where ν is the kinematic viscosity and $\delta_c(x)$ is a length scale proportional to the boundary layer thickness, $\delta_c(x) = \sqrt{\nu x/U_e}$, where U_e is the unperturbed boundary layer edge velocity. The length scale, $\delta_c(x)$, is typically used in PSE analysis [58, 5]. The reference length scale used to nondimensionalize the equations is δ_c evaluated at the first marching streamwise coordinate [65]. The wall-normal component, $\bar{v}$, is nonzero and scales with $1/Re$. Formally defining the slowly varying scale $x^* = x/Re_{\delta_c}$, the scalings are:

$$\begin{aligned} \bar{w} &\sim \frac{1}{Re_{\delta_c}}, \\ \frac{\partial}{\partial x^*} &\sim \frac{1}{Re_{\delta_c}}, \\ \alpha &= \alpha\left(x^*\right), \\ \hat{\mathbf{q}} &= \hat{\mathbf{q}}\left(x^*, y\right). \end{aligned} \tag{4.9}$$

In a linear PSE analysis, the perturbation vector is expanded in terms of a single mode, truncated Fourier component assuming time-periodicity,

$$\tilde{\mathbf{q}}(x, y, x, t) = \hat{\mathbf{q}}(x, y) \exp\left[i\left(\int_x \alpha\left(x'\right) dx' + \beta z - \omega t\right)\right], \tag{4.10}$$

where $\hat{\mathbf{q}}(x, y)$ has a slow variation in x. The flow disturbance amplitudes, $\hat{\mathbf{q}}$, present the three velocity components even when a two-dimensional base flow is considered.

To obtain the linear PSE equations, we replace the flow state vector decomposition, $\mathbf{q}(\mathbf{x}, t) = \bar{\mathbf{q}}(\mathbf{x}) + \epsilon \tilde{\mathbf{q}}(\mathbf{x}, t)$, in the linearized Navier–Stokes equations and neglect terms of $\mathcal{O}(\epsilon^2)$. We also consider the scaling from Eq. (4.9) and neglect higher derivatives with

respect to x in the viscous terms (noting that $\frac{\partial}{\partial x}\frac{1}{Re_{\delta_c}} \sim \epsilon^2$). The resulting linear PSE equations, in compact form, read

$$\mathbf{A}\hat{\mathbf{q}} + \mathbf{B}\frac{1}{h_y}\frac{\partial \hat{\mathbf{q}}}{\partial y} + \mathbf{C}\frac{1}{h_y^2}\frac{\partial^2 \hat{\mathbf{q}}}{\partial y^2} + \mathbf{D}\frac{1}{h_x}\frac{\partial \hat{\mathbf{q}}}{\partial x} = \mathbf{0}, \tag{4.11}$$

where h_x and h_y are curvature metrics. The entries for the compressible PSE operators $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are detailed by Hanifi et al. [58]. The boundary conditions are:

$$\hat{u} = \hat{v} = \hat{w} = 0 \quad \text{at} \quad y = 0, \tag{4.12}$$

$$\hat{u} = \hat{v} = \hat{w} = 0 \quad \text{as} \quad y \to \infty. \tag{4.13}$$

In the PSE framework, changes in amplitude along the slowly varying spatial direction are contained both in the amplitude function $\hat{\mathbf{q}}$ and in the phase function defined in Eq. (4.10). To remove such ambiguity, a normalization condition is required. One possibility for the normalization is to impose that the total kinetic energy of the shape function be independent of x so that fast disturbance variations in the streamwise direction are absorbed into the phase function [4],

$$\int_0^\infty \hat{\mathbf{u}}^\dagger \frac{\partial \hat{\mathbf{u}}}{\partial x} dy = 0, \tag{4.14}$$

where the superscript † represents the complex conjugate. This normalization condition also ensures an assumed scaling of $\partial\hat{\mathbf{q}}/\partial x$ with $1/Re$. The disturbance kinetic energy is used to measure the disturbance growth,

$$E(x) = \frac{1}{2}\int_0^\infty \left(|\hat{\mathbf{u}}|^2 + |\hat{\mathbf{v}}|^2 + |\hat{\mathbf{w}}|^2\right) dy. \tag{4.15}$$

The disturbance kinetic energy-based disturbance growth ratio reads

$$\sigma_E = -\alpha_i + \frac{d}{dx}\ln\sqrt{E(x)}. \tag{4.16}$$

To compute the onset of the transition region, the e^N method is usually employed, where N is the amplification factor, or N-factor, defined as

$$N = \ln\left(\frac{A}{A_0}\right) = \int_{x_0}^{x_n} \sigma_E(x)\, dx, \tag{4.17}$$

where A_0 is the disturbance amplitude at the first neutral-stability point location, denoted by x_0, and x_n the final station. The N-factor envelope is obtained as the superposition of different N-factor curves at each station, computed by the PSE code for a range of frequencies and spanwise wavenumbers.

Transition to turbulence is assumed to occur when the N-factor reaches a critical value, N_{crit}. For Tollmien–Schlichting waves, this critical threshold is given by Mack's correlation [56]:

$$N_{crit,TS} = -8.43 - 2.4\ln(Tu), \tag{4.18}$$

where Tu is the turbulence intensity. This correlation is valid within the range $0.001 < Tu < 0.01$.

The linear PSE (4.11) are intended to be parabolic. Therefore, it is possible to treat the streamwise direction as a pseudo-time and then to implement a marching strategy in this spatial direction. Numerical instabilities appear when the streamwise integration step is too small [5]. The reason for that, as explained by Herbert [4], is that there are traces of ellipticity that inject ill-posed characteristics. One remedy for this is the use of a first-order backward difference scheme with a lower integration step limit $\Delta x > 1/|\alpha_r|$. To relax this limit, Andersson et. al. [67] propose a stabilization procedure leading to $\Delta x > 1/|\alpha_r| - 2s$, where s is a small number.

Due to the predominantly parabolic character of the linear PSE (4.11), the disturbance evolution is influenced by both local and upstream flow information and, therefore, the PSE method is recognized as a *nonlocal* approach, in contrast to, for instance, the Orr–Sommerfeld equation, which is a *local* formulation.

4.2 PSE solver implementation details

We use the incompressible PSE flow solver introduced in Ref. [5]. Spectral collocation points are used for the wall-normal discretization of Eq. (4.11). The Chebyshev nodes are defined as follows,

$$y_{\mathrm{cheb},i} = \cos\left(\frac{\pi i}{N_y}\right), \tag{4.19}$$

where N_y is the number of points in the wall-normal direction. A Chebyshev spectral differentiation matrix can be used to compute the derivative of vector $\hat{\mathbf{q}}$ through a matrix-vector product,

$$\frac{d\hat{\mathbf{q}}}{d\mathbf{y}} = \mathbf{D}_{1,\mathrm{cheb}}\hat{\mathbf{q}}^T. \tag{4.20}$$

The entries for the Chebyshev spectral differentiation matrix read [68],

$$\left(\mathbf{D}_{1,\mathrm{cheb}}\right)_{i,j} = \begin{cases} \frac{2N_y^2+1}{6} & \text{for} \quad i=j=0, \\ -\frac{2N_y^2+1}{6} & \text{for} \quad i=j=N_y, \\ \frac{-y_{\mathrm{cheb},i}}{2\left(1-y_{\mathrm{cheb},i}^2\right)} & \text{for} \quad i=j, \quad 1 \le j \le N_y-1, \\ \frac{c_i}{c_j}\frac{(-1)^{i+j}}{y_{\mathrm{cheb},i}-y_{\mathrm{cheb},j}} & \text{for} \quad i \neq j, \end{cases} \tag{4.21}$$

where

$$c_i = \begin{cases} 2 & \text{for} \quad i=0 \quad \text{or} \quad i=N_y, \\ 1 & \text{for} \quad 1 \le i \le N_y-1. \end{cases} \tag{4.22}$$

The second-derivative Chebyshev differentiation matrix is simply

$$\mathbf{D}_{2,\mathrm{cheb}} = \mathbf{D}_{1,\mathrm{cheb}}\mathbf{D}_{1,\mathrm{cheb}}. \tag{4.23}$$

A mapping to the physical domain, y_{phys}, is used to cluster points close to the wall to capture the wave mode activity in the near-wall region. Suitable mappings between the Chebyshev and physical spaces for incompressible and compressible flows are available in the literature [69, 70]. The mapping between the spectral and physical spaces reads, for subsonic flows,

$$y_{\text{phys},i} = a\left(\frac{1 + y_{\text{cheb},i}}{b - y_{\text{cheb},i}}\right). \tag{4.24}$$

The coefficients a and b are,

$$a = \frac{y_h y_{\max}}{y_{\max} - 2y_h}. \tag{4.25}$$

where y_h denotes the wall-normal coordinate up to which half of the nodes are distributed in the case of low Mach number (M). For $M > 0.5$, y_h indicates the position away from the wall where a fraction c of the total number of points are located. In our implementation, we use $y_h = 0.4$ and $c = 0.4$. Term b in Eq. (4.25) is defined as,

$$b = 1 + \left(\frac{2a}{y_{\max}}\right). \tag{4.26}$$

The spectral differentiation matrix is also mapped to the physical space,

$$(\mathbf{D}_{1,\text{phys}}) = (\mathbf{D}_{1,\text{cheb}})\,\frac{dy_{\text{phys}}}{dy_{\text{cheb}}}. \tag{4.27}$$

For the second-derivative matrix,

$$(\mathbf{D}_{2,\text{phys}}) = (\mathbf{D}_{1,\text{cheb}})\,\frac{d^2 y_{\text{phys}}}{dy_{\text{cheb}}^2} + (\mathbf{D}_{2,\text{cheb}})\left(\frac{dy_{\text{phys}}}{dy_{\text{cheb}}}\right)^2. \tag{4.28}$$

The semi-discrete counterpart of Eq. (4.11) becomes,

$$\mathbf{A}\hat{\mathbf{q}} + \frac{1}{h_y}\mathbf{B}\mathbf{D}_{1,\text{phys}}\hat{\mathbf{q}} + \frac{1}{h_y^2}\mathbf{C}\mathbf{D}_{2,\text{phys}}\hat{\mathbf{q}} + \mathbf{D}\frac{1}{h_x}\frac{\partial\hat{\mathbf{q}}}{\partial x} = \mathbf{0}. \tag{4.29}$$

A fully-discrete version of Eq. (4.29) is obtained by replacing the streamwise coordinate derivative with a first or second-order backwards Euler marching scheme, in a way that x can be interpreted as a pseudo time.

To determine the initial solution, two approaches are available. It is possible to use an LST method to compute the initial solution after the Gaster transformation is applied to the temporal problem, leading to a solution in the spatial framework. Another approach modifies the original PSE equations to yield a local, parallel problem that is solved in the initial marching station in a way that a previous LST solution is not used. In this case, we modify the PSE operators defined in Eq. (4.11) to reflect the nature of a parallel problem. To do this, derivatives in the streamwise direction are excluded and the base flow velocity in this direction is set to zero. To obtain a local problem, meaning that the flow stability

problem in each station is independent of the previous ones, we set the $\mathbf{D}$ matrix to zero, and Eq. (4.11) becomes:

$$\mathbf{A}_{p,l}\hat{\mathbf{q}} + \mathbf{B}_{p,l}\frac{1}{h_y}\frac{\partial \hat{\mathbf{q}}}{\partial y} + \mathbf{C}_{p,l}\frac{1}{h_y^2}\frac{\partial^2 \hat{\mathbf{q}}}{\partial y^2} = \mathbf{0}, \tag{4.30}$$

where the subscript p, l indicates the local, parallel solution obtained through the modified PSE equation. The discrete counterpart of Eq. (4.30) is,

$$\mathbf{A}_{p,l}\hat{\mathbf{q}}_1 + \frac{1}{h_y}\mathbf{B}_{p,l}\mathbf{D}_{1,\mathrm{phys}}\hat{\mathbf{q}}_1 + \frac{1}{h_y^2}\mathbf{C}_{p,l}\mathbf{D}_{2,\mathrm{phys}}\hat{\mathbf{q}}_1 = \mathbf{0}. \tag{4.31}$$

We solve Eq. (4.31) iteratively in an inexpensive way to avoid the computation of the entire eigenmode space that results from the solution of the local, parallel problem through a standard eigenvalue solver. Further details regarding the eigenmode initialization procedure can be found in Ref. [6]. The steps followed in the PSE calculation are summarized in Algorithm 3, adapted from Ref. [6].

Algorithm 3: PSE computation

$\hat{\mathbf{q}}_1 \leftarrow \mathbf{0}$ ▷ `Initialize local solution`

$\alpha_1 \leftarrow \alpha_0$ ▷ `Initialize local wavenumber`

while $\hat{v}_{\text{wall}} > 2 \times 10^{-8}$ **do**

$\mathbf{A}_{p,l}\hat{\mathbf{q}}_1 + \frac{1}{h_y}\mathbf{B}_{p,l}\mathbf{D}_{1,\text{phys}}\hat{\mathbf{q}}_1 + \frac{1}{h_y^2}\mathbf{C}_{p,l}\mathbf{D}_{2,\text{phys}}\hat{\mathbf{q}}_1 = \mathbf{0}$ ▷ `Solve the local problem`

$\zeta \leftarrow \hat{w}_{\text{wall}}$

$\alpha_n \leftarrow \alpha_{n-1} - \zeta(\alpha_{n-1})\frac{\alpha_{n-1} - \alpha_{n-2}}{\zeta(\alpha_{n-1}) - \zeta(\alpha_{n-2})}$ ▷ `Wavenumber update`

$\Gamma \leftarrow 10^6$ ▷ `Initialize normalization condition to a large number`

for i ***in*** n_{stations} **do**

while $\Gamma > 10^{-8}$ **do**

$\mathbf{A}\hat{\mathbf{q}} + \frac{1}{h_y}\mathbf{B}\mathbf{D}_{1,\text{phys}}\hat{\mathbf{q}} + \frac{1}{h_y^2}\mathbf{C}\mathbf{D}_{2,\text{phys}}\hat{\mathbf{q}} + \mathbf{D}\,\frac{1}{h_x}\frac{\partial \hat{\mathbf{q}}}{\partial x} = \mathbf{0}$ ▷ `Solve PSE problem`

$\Gamma \leftarrow \int_0^\infty \hat{\mathbf{u}}^\dagger \frac{\partial \hat{\mathbf{u}}}{\partial x}\, dy$ ▷ `Compute the normalization condition`

$\zeta \leftarrow \Gamma$

$\alpha_n \leftarrow \alpha_{n-1} - \zeta(\alpha_{n-1})\frac{\alpha_{n-1} - \alpha_{n-2}}{\zeta(\alpha_{n-1}) - \zeta(\alpha_{n-2})}$ ▷ `Wavenumber update`

$\sigma \leftarrow \frac{1}{h_x}\left(-\alpha_i + \Re\left[\frac{1}{\xi}\frac{\partial \xi}{\partial x}\right]\right)$ ▷ `Compute the disturbance growth rate`

$N \leftarrow \int_{x_0}^{x} \sigma(x')\, dx'$ ▷ `Compute` N`-factor`

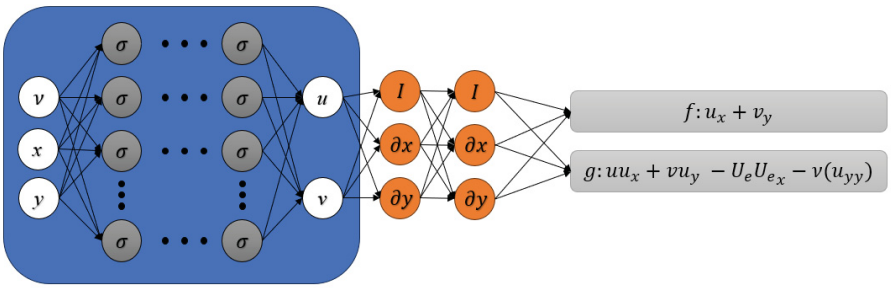
5 PINN reconstruction

This chapter details the physics-informed neural network (PINN) formulation employed in this work. The formulation is first applied to the flat plate laminar boundary layer problem and subsequently to flow-stability analysis based on the Parabolized Stability Equations (PSE).

5.1 PINN formulation for boundary layer flows

Using the concept of PINN and considering a two-dimensional boundary layer, we approximate the solution of the laminar boundary layer equations with a deep-neural network to predict (u, v) for a given (ν, x, y) set, where ν is the kinematic viscosity. Here, x and y represent the domain coordinates in the streamwise and wall-normal directions, respectively. Figure 6, which was adapted from Ref. [13], describes the structure of the PINN used in this work, which consists of a fully-connected network and the residual networks. The activation function, σ, is select to be the hyperbolic tangent function, *tanh*.

Figure 5 – Physics-informed neural network for an incompressible laminar boundary layer flow.



Source: The author (2025).

To address the phenomenon of interest, the boundary layer momentum equations can be written as,

$$uu_x + vu_y = U_e U_{ex} + \nu(u_{yy}), \tag{5.1}$$

$$p_y = 0, \tag{5.2}$$

where subscripts indicate differentiation. Here, since we are dealing with flow without a pressure gradient, the term $U_e U_{ex}$ will be equal to zero.

The equation of mass conservation for incompressible boundary layer flows reads

$$u_x + v_y = 0. \tag{5.3}$$

We assume that a latent function $\psi(x, y)$ exists such that,

$$u = \psi_y, \quad v = -\psi_x. \tag{5.4}$$

When a latent function ψ exists and Eq. (5.3) is satisfied, even if noise measurements occur, the continuity equation is met.

Given noisy measurements of the velocity field expressed by,

$$\{x^i, y^i, u^i, v^i\}_{i=1}^N, \tag{5.5}$$

we are interested in learning the velocity field $u(x, y)$ and $v(x, y)$. We then define $f(x, y)$ and $g(x, y)$,

$$f := u_x + v_y, \tag{5.6}$$

$$g := uu_x + vu_y - U_e U_{ex} - \nu(u_{yy}), \tag{5.7}$$

Eqns. (5.6) and (5.7), along with the assumptions made above, result in the physics-informed neural network, which can be trained by minimizing the mean squared error loss,

$$MSE := \frac{1}{N}\sum_{i=1}^{N}(|u(x^i, y^i) - u^i|^2 + |v(x^i, y^i) - v^i|^2) + \frac{1}{N}\sum_{i=1}^{N}(|f(x^i, y^i)|^2 + |g(x^i, y^i)|^2). \tag{5.8}$$

To train the neural network, we initially use SciANN [45], a library specifically designed for physics-informed neural networks (PINNs). The network architecture and training parameters were meticulously defined to ensure accurate learning. Specifically, we use the ADAM optimizer [39], a popular first-order gradient-based optimization algorithm, to minimize the error in the loss function. The learning rate is set to 1×10^{-4} to balance convergence speed and training stability, and the activation function used for all of our results is the hyperbolic tangent. The training dataset obtained from SciANN comprises velocity profiles, including both the u and v velocity fields, generated by the boundary layer code. After the data filtering mentioned above, the dataset consists of 45,232 points. In our implementation of PINNs for boundary layer flows in TensorFlow, we use 20,000 data points without applying any filtering. These data points serve as inputs and targets for the neural network.

5.2 PINN formulation for PSE

In this work, is consider a two-dimensional, incompressible base flow over a flat plate. Considering the incompressible PSE operators **A**, **B**, **C**, and **D**, given by [5]:

$$\mathbf{A} = \begin{bmatrix} i\alpha & 0 & i\beta & 0 \\ \zeta + \frac{\partial U}{\partial x} & \frac{\partial U}{\partial y} & 0 & i\alpha \\ 0 & \zeta + \frac{\partial V}{\partial y} & 0 & 0 \\ \frac{\partial W}{\partial x} & \frac{\partial W}{\partial y} & \zeta & i\beta \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ V & 0 & 0 & 0 \\ 0 & V & 0 & 1 \\ 0 & 0 & V & 0 \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -\frac{1}{Re_{\delta_c}} & 0 & 0 & 0 \\ 0 & -\frac{1}{Re_{\delta_c}} & 0 & 0 \\ 0 & 0 & -\frac{1}{Re_{\delta_c}} & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ U & 0 & 0 & 1 \\ 0 & U & 0 & 0 \\ 0 & 0 & U & 0 \end{bmatrix}$$

where

$$\zeta = -i\omega + i\alpha U + i\beta W + \frac{1}{Re_{\delta_c}}(\alpha^2 + \beta^2), \tag{5.9}$$

and assuming unitary curvature metrics, $h_x = h_y = 1$, Eq. (4.11) are written as:

$$i\alpha\hat{u} + i\beta\hat{w} + \frac{\partial\hat{u}}{\partial x} + \frac{\partial\hat{v}}{\partial y} = 0, \tag{5.10}$$

$$\left(\zeta + \frac{\partial U}{\partial x}\right)\hat{u} + \frac{\partial U}{\partial y}\hat{v} + i\alpha\hat{p} + V\frac{\partial\hat{u}}{\partial y} - \frac{1}{Re}\frac{\partial^2\hat{u}}{\partial y^2} + U\frac{\partial\hat{u}}{\partial x} + \frac{\partial\hat{p}}{\partial x} = 0, \tag{5.11}$$

$$\left(\zeta + \frac{\partial V}{\partial y}\right)\hat{v} + V\frac{\partial\hat{u}}{\partial y} - \frac{1}{Re}\frac{\partial^2\hat{v}}{\partial y^2} + U\frac{\partial\hat{v}}{\partial x} + \frac{\partial\hat{p}}{\partial y} = 0, \tag{5.12}$$

$$\frac{\partial W}{\partial x}\hat{u} + \frac{\partial W}{\partial y}\hat{v} + \zeta\hat{w} + i\beta\hat{p} + V\frac{\partial\hat{w}}{\partial y} - \frac{1}{Re}\frac{\partial^2\hat{w}}{\partial y^2} + U\frac{\partial\hat{w}}{\partial x} = 0. \tag{5.13}$$

In Eqs. (5.11) to (5.9), U, V, and W are the base flow velocity components in the x, y, and z directions, respectively, i is the imaginary unit, α is the streamwise wave number, ω is the angular frequency, and β is the spanwise wave number.

Using the concept of PINN and considering a PSE-based flow stability analysis, we approximate the PSE solution with a deep-neural network to predict $(\hat{u}, \hat{v}, \hat{w}, \hat{p}, \alpha)$ for a given $(x, y, Re, \omega, \beta, U, V, W, U_x, U_y, V_y, W_x, W_y)$ set, where subscripts denote differentiation. Figure 6, which was adapted from Ref. [13], describes the structure of the PINN used in this work, which consists of a fully-connected network and the residual networks. The activation function σ is the hyperbolic tangent function, *tanh*.

We rewrite the PSE equations as:

$$i\alpha\hat{u} + i\beta\hat{w} + \hat{u}_x + \hat{v}_y = 0, \tag{5.14}$$

$$(\zeta + U_x)\hat{u} + U_y\hat{v} + i\alpha\hat{p} + V\hat{u}_y - Re^{-1}\hat{u}_{yy} + U\hat{u}_x + \hat{p}_x = 0, \tag{5.15}$$

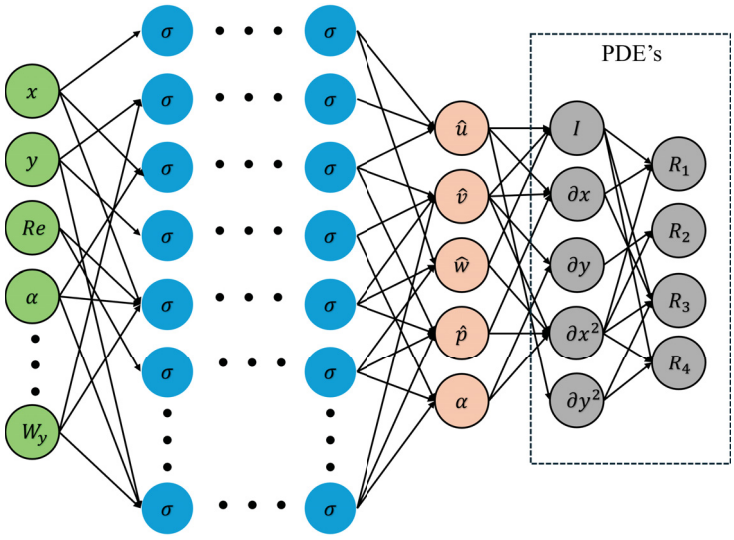


Figure 6 – Physics-informed neural network for parabolized stablity equations.

$$(\zeta + V_y)\hat{v} + V\hat{u}_y - Re^{-1}\hat{v}_{yy} + U\hat{v}_x + \hat{p}_y = 0, \tag{5.16}$$

$$W_x\hat{u} + W_y\hat{v} + \zeta\hat{w} + i\beta\hat{p} + V\hat{w}_y - Re^{-1}\hat{w}_{yy} + U\hat{w}_x = 0, \tag{5.17}$$

where subscripts indicate differentiation. The base flow velocity gradients, U_x, U_y, V_y, W_x, W_y, are directly computed within the PSE flow solver [5]. These base flow velocity gradients are then directly fed into a neural network. We are interested in learning the disturbance's amplitude field $\hat{u}(x, y)$, $\hat{v}(x, y)$, and $\hat{w}(x, y)$, as well as the streamwise wavenumber, $\alpha(x)$. We then define the residual equations, R_1, R_2, R_3 and R_4,

$$R_1 := i\alpha\hat{u} + i\beta\hat{w} + \hat{u}_x + \hat{v}_y, \tag{5.18}$$

$$R_2 := (\zeta + U_x)\hat{u} + U_y\hat{v} + i\alpha\hat{p} + V\hat{u}_y - Re^{-1}\hat{u}_{yy} + U\hat{u}_x + \hat{p}_x, \tag{5.19}$$

$$R_3 := (\zeta + V_y)\hat{v} + V\hat{u}_y - Re^{-1}\hat{v}_{yy} + U\hat{v}_x + \hat{p}_y, \tag{5.20}$$

$$R_4 := W_x\hat{u} + W_y\hat{v} + \zeta\hat{w} + i\beta\hat{p} + V\hat{w}_y - Re^{-1}\hat{w}_{yy} + U\hat{w}_x. \tag{5.21}$$

As previously described, the perturbation fields $(\hat{u}, \hat{v}, \hat{w}, \hat{p})$ and the streamwise wavenumber α are complex-valued quantities. To handle these variables within the neural network framework, each partial differential equation that is part of the PSE system is decomposed into real and imaginary parts. Specifically, the residuals of the PSE, R_1, R_2, R_3, and R_4, are split into:

$$R_k = \mathrm{Re}(R_k) + i\,\mathrm{Im}(R_k), \quad k = 1, 2, 3, 4. \tag{5.22}$$

This allows the network to predict both real and imaginary parts of the perturbation fields independently while ensuring that the full complex equations are satisfied. During training, the loss function accounts for both components simultaneously, minimizing the squared residuals of the real and imaginary parts, which enforces the physics embedded in the PSE on the neural network solution in a consistent manner.

Equations (5.18), (5.19), (5.20) and (5.21), along with the assumptions made above, result in the physics-informed neural network mathematical formulation, which can be trained by minimizing the mean squared error loss,

$$MSE := \frac{1}{N_d}\sum_{k=1}^{N_d}(|\hat{u}^k_{ref} - \hat{u}^k_{pred}|^2 + |\hat{v}^k_{ref} - \hat{v}^k_{pred}|^2 + |\hat{w}^k_{ref} - \hat{w}^k_{pred}|^2 + |\hat{p}^k_{ref} - \hat{p}^k_{pred}|^2 + |\alpha^k_{ref} - \alpha^k_{pred}|^2) + \frac{1}{N_c}\sum_{k=1}^{N_c}(|R^k_1|^2 + |R^k_2|^2 + |R^k_3|^2 + |R^k_4|^2), \tag{5.23}$$

where the subscripts d and c indicate the training data and collocation points, respectively. To ensure that the neural network solution obeys the governing physical laws, even in regions where no training data are available, the concept of *collocation points* is used. These are coordinate points sampled throughout the problem domain at which the exact solution is unknown. The purpose of these points is to force the network to satisfy the Parabolized Stability Equations (PSE). In practice, for each collocation point, the neural network output is used to compute the residuals of the PDEs, Eqs. (5.18) to (5.21). These residuals form the second term of the loss function, Eq. (5.23), which is minimized during training together with the data error. This process acts as a strong physics-based regularizer, enabling the model to learn a physically consistent solution and to generalize well, even from a sparse training dataset, a capability demonstrated in the results of Section 6.3.2.

6 Numerical results

In this chapter, incompressible flat plate boundary layer flows with zero pressure gradient are investigated. Both standard and physics-informed neural networks are trained and assessed for their ability to recover boundary layer velocity profiles. Flow stability analysis based on a nonlocal, nonparallel approach is also considered, comparing the performance of PINNs with the results obtained from standard feedforward neural networks in a sparse data scenario.

6.1 Results: Boundary layer flow prediction with SciANN

Flat plate boundary layer investigations are initiated based on training performed using SciANN. Results obtained with the trained models are compared with those obtained directly from the boundary layer solver.

6.1.1 Data generation

To generate the training dataset for the neural network in SciANN, we focus on flat plate boundary layers with zero pressure gradient. The flow is assumed to be laminar and subject to a uniform free stream velocity of $U_\infty = 1.00$ m/s, with a Reynolds number based on the flat plate length of 250,000. The computational domain of the boundary layer code extends from $x = 0.25$ m to $x = 1.00$ m, with a grid of 300 points in both x- and y-directions to ensure a high-quality solution. The dataset used for the neural network training consists of both u and v velocity fields, covering the length from $x = 0.50$ m to $x = 1.00$ m. A manipulation of the dataset is applied to ensure effective training of the neural network. This manipulation is divided into two main steps. The first step consists of removing data points where the velocity component u is greater than 0.9999, which corresponds to the value of the velocity outside the boundary layer region, where the flow is uniform and $u \approx 1$. The motivation for this removal is to avoid redundancy, since in the free-stream region all points carry the same information. Keeping these points does not provide any new learning for the neural network and unnecessarily increases the computational cost of training. This filtering ensures that the network focuses only on the region of interest, where velocity variations occur. Another motivation for this filtering is to avoid zero gradients during training. Neural networks learn by computing gradients, i.e., rates of change in the data. In regions where the flow is constant, the gradient is effectively zero, meaning the network cannot extract useful error information, which hinders the weight update process and slows down training, making it inefficient. The second step of the manipulation involves scaling the velocity component v by a factor of 300. This

step aims to balance the magnitudes of the velocity components. In a flow over a flat plate, the wall-normal velocity component v has a significantly smaller magnitude than the streamwise component u. This imbalance causes the neural network to give much more importance to variations in u at the expense of v, simply because u has a much larger magnitude. By multiplying v by 300, its value is amplified, approaching the magnitude of u, thereby balancing the relative importance of both components during training. The factor magnitude was chosen based on the theoretical relation $v \sim u/\sqrt{Re}$. The steps described above ensure that the network is fed with clean, well-normalized data, promoting proper behavior and efficient learning during training. Since the flow develops mainly in the x-direction, the velocity values in the y-direction are much smaller than the u-velocity. This scaling improves the efficiency of the optimization algorithm by producing more uniform gradients in both velocity fields. The testing dataset spans from $x = 0.25$ m to $x = 1.00$ m. The error estimates presented here are based on streamwise positions ranging from $x = 0.50$ m to $x = 1.00$ m, while the velocity field reconstructions presented later in this section consider positions from $x = 0.25$ m to $x = 1.00$ m.

6.1.2 Results: boundary layer prediction with SciANN

To determine the optimal configuration that minimizes error, we test various combinations of numbers of layers and neurons, as shown in Table 2. The simulations presented here were performed on a machine equipped with a ninth-generation Intel i5-9400F processor, featuring 6 cores and 6 threads, operating at a frequency of 4.10 GHz. The system also has 16 GB of DDR4-2666 RAM. By varying the network architectures, we observe the behavior of errors in the u and v velocity fields, as well as the computational time required to reach a loss function threshold of 1×10^{-4}. The prediction errors for the velocity components u and v are quantified using the L^2 norm. For this initial training, we use a total of 44,000 data points, and the remaining 1,232 points are used for testing the trained models. The testing set corresponds to streamwise locations beyond the mid-chord position, where $x = 0.50$ m. Inspection of Table 2 reveals that 40 layers and 40 neurons per

Table 2 – Comparison of errors between PINN and standard neural network for different architectures.

Layers	**Neurons**	**PINN**			**Standard N. N.**		
		Error u (%)	**Error v (%)**	**time (s)**	**Error u (%)**	**Error v (%)**	**time (s)**
30	30	1.9320	2.3219	4078	0.9721	0.6199	1985
40	40	0.2491	0.7063	16748	0.4690	0.7469	1630
50	50	0.9458	0.8868	29154	1.0176	0.5083	4590

layer represent the architecture that leads to minimum errors in the u velocity component for both standard and physics-informed neural networks. We select this architecture for our next investigations because, in agreement with boundary layer theory, the u velocity component is expected to present larger magnitudes and gradients than the wall-normal

velocity component, v. This configuration significantly reduced the error for both fields, leading to 0.2491% and 0.4690% errors in u for the PINN and the standard neural network, respectively. However, it is worth mentioning that the training time for the PINN is substantially higher compared to the one corresponding to the standard neural network. The increased computational overhead of evaluating PDE residuals, imposing physical constraints, and calculating derivatives makes the training process for PINNs more time-intensive than for standard neural networks for the same network architecture and training set. Nonetheless, this additional time is often justified by the superior generalization and adherence to physical laws offered by PINNs, as discussed later in this section.

Both neural networks are also compared, standard and physics-informed, by varying the dataset sizes while maintaining the same number of neurons and layers for both models, as shown in Table 3:

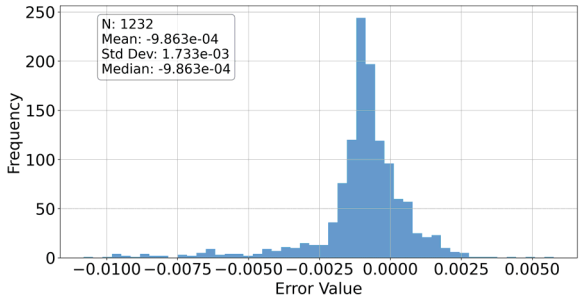
Table 3 – Comparison of errors between PINN and standard neural networks for different training datasets.

Cases	**Train. data**	**Test data**	**PINN**			**Standard N. N.**		
			Error u (%)	**Error v (%)**	**time (s)**	**Error u (%)**	**Error v (%)**	**time (s)**
Case 1	44,000	1,232	0.2491	0.7063	16,748	0.4690	0.7469	1,630
Case 2	22,000	23,232	0.7018	0.7175	5,453	1.0367	0.7174	2,948
Case 3	11,000	34,232	0.6412	1.2744	5,116	0.8291	0.8563	1,950

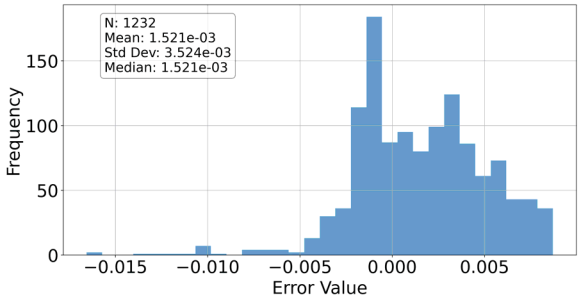
The incorporation of physical knowledge through differential equations reduces errors in both the u and v fields. This is especially evident when the number of training data points is limited, demonstrating the superior generalization capability of PINNs under data-scarce conditions. For instance, in case 2, with only 22,000 data points for training, the PINN achieves substantially lower errors compared to the standard neural network. In this specific case, the error in the u velocity component is as low as 0.7018% for the PINN, while the standard neural network delivers an error, for the same velocity component, of 1.0367%. For the reduced training data sets, comprising 22,000 and 11,000 data points, the PINN training times are of the same order as those for the standard neural network, although still higher for the physics-informed approach.

Considering all three cases, with training data sets from 11 to 44 thousand points, we can perform an error analysis for both the PINN and the standard neural network, as seen in Fig. 7. This error is calculated by computing the difference between the test data and the predicted data, $u(x^i, y^i) - u^i$ and $v(x^i, y^i) - v^i$, for the same points. Both the test and training data are sourced from the region starting at $x = 0.50$ m to the end of the flat plate, which is located at $x = 1.00$ m. Inspection of Figure 7 indicates that, for the training set composed of 11,000 data points, both mean and median are smaller for the PINN than for the standard neural network. The opposite is observed for a training set with 22,000 data points, with equivalent error mean and median for the PINN and the

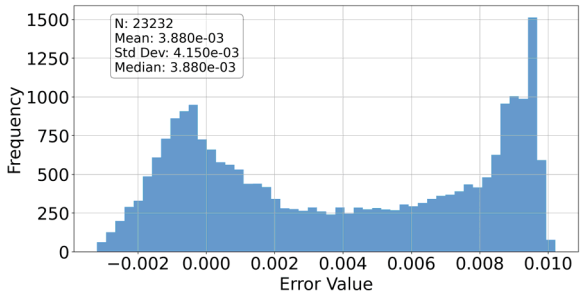
Figure 7 – Error histograms for the streamwise velocity component, u, for varying training data set sizes.



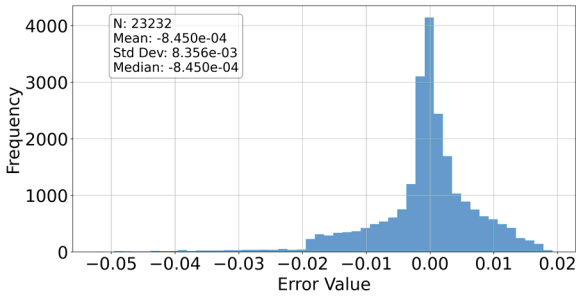
(a) PINN - Case 1



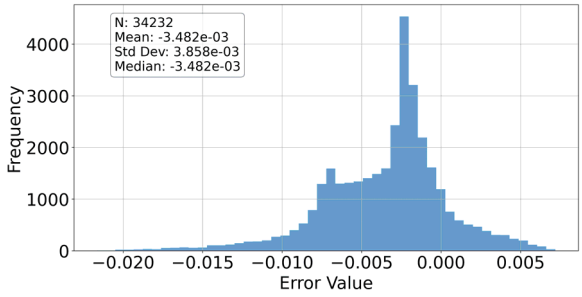
(b) Standard neural network - Case 1



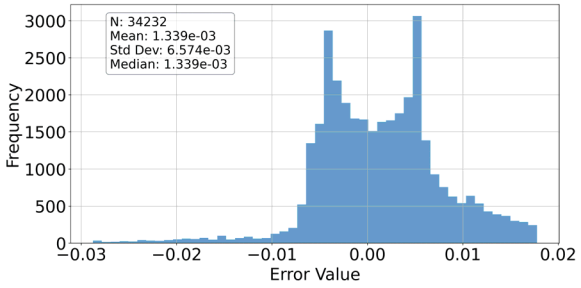
(c) PINN - Case 2



(d) Standard neural network - Case 2



(e) PINN - Case 3



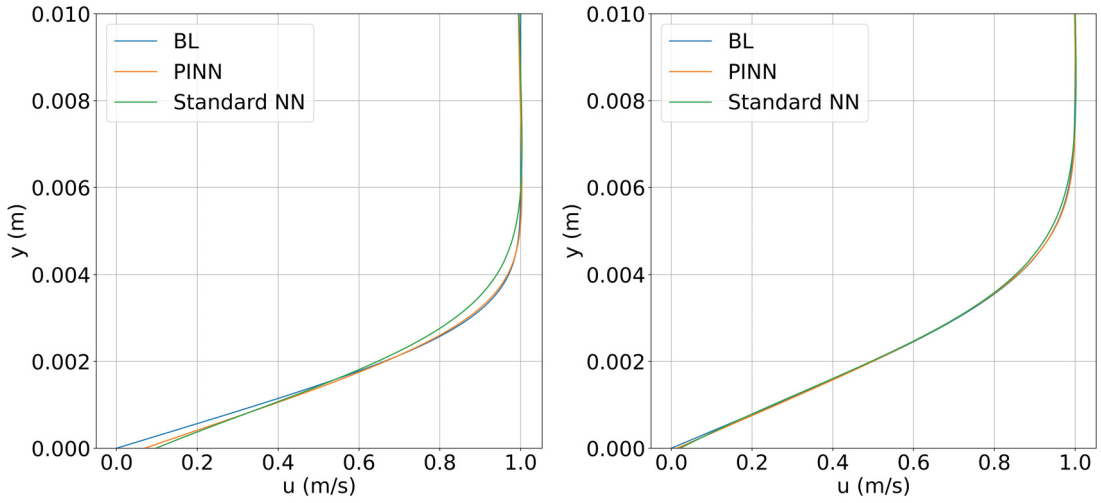
(f) Standard neural network - Case 3

standard neural network when the training set has 44,000 points.

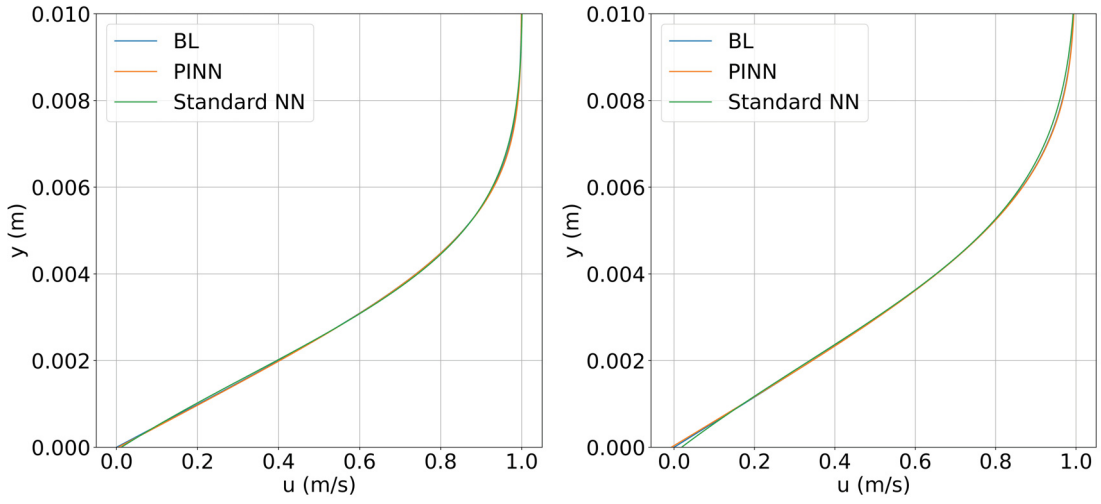
Figures 8 to 10 present detailed comparisons of the velocity profiles at four different streamwise positions, namely, $x = 0.30, 0.50, 0.75$, and 1.00 m. The u velocity component profiles as obtained by the boundary layer solver and as predicted by both the standard and the physics-informed neural networks are compared. For Case 1, which considers 44,000 data points for training, an analysis of Figure 8a indicates that, in the region predicted without reference data, both predictions, PINN and Standard NN, exhibit small deviations from the reference solution obtained from the boundary layer solver. Near the wall, there is a discrepancy for both networks. However, in the region close to the edge of the boundary layer, it can be observed that the standard neural network struggles to predict the velocity component u, whereas the PINN demonstrates higher accuracy with respect to the reference solution. For the remaining regions shown in Figs. 8b, 8c, 8d, it is possible to observe that both networks provide predictions equivalent to the reference

solution, with small variations favoring the PINN. For instance, at $x = 1.00$ m, a minor prediction error by the standard neural network can be observed near the wall. This superior performance highlights the advantage of incorporating physical laws into the neural network, allowing for more accurate and reliable predictions, especially in critical regions where precise modeling is essential. This is relevant when performing flow stability computations, for which the quality of the velocity profiles and gradients directly impacts the accuracy of the computations.

Figure 8 – Streamwise velocity component, u, profiles as predicted by the boundary layer solver and both standard and physics-informed neural networks for 44,000 training data points (case 1).



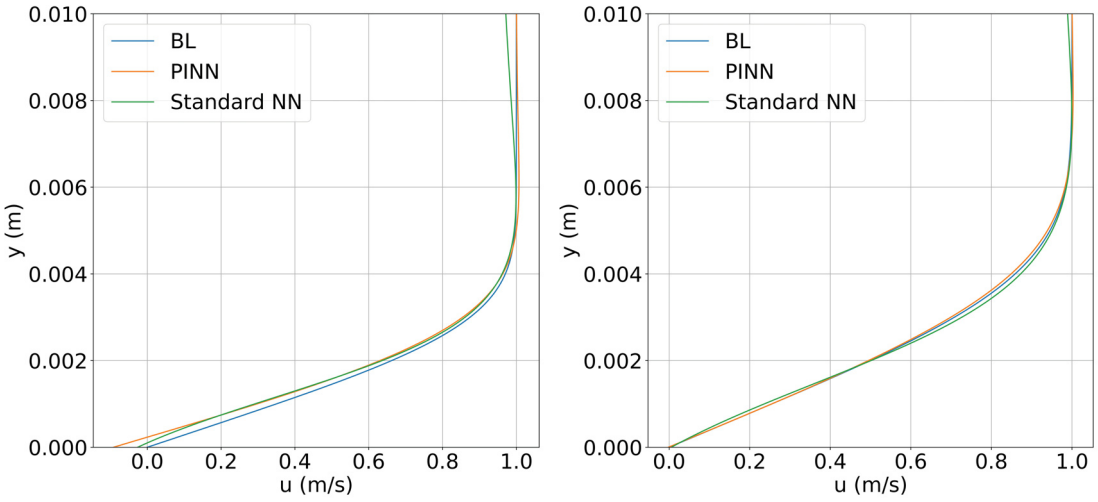
(a) $x = 0.30$ m (b) $x = 0.50$ m



(c) $x = 0.75$ m (d) $x = 1.00$ m

The velocity profiles at the selected streamwise locations for case 2, which is based on a training data set with 22,000 points, are inspected in Fig. 9. For this case, a behavior similar to the previous one can be observed. At the location $x = 0.30$ m, both networks encounter difficulties in predicting the velocity component in the streamwise direction. Near the wall, prediction errors are present in both the PINN and the Standard Neural Network (SNN). However, in the region close to the boundary layer edge, the PINN once again provides a significantly more accurate prediction compared to the standard neural network. In Figures 9c and 9d, the difficulties of the standard neural network are evident, especially in predicting the flow close to the wall.

Figure 9 – Streamwise velocity component, u, profiles as predicted by the boundary layer solver and both standard and physics-informed neural networks for for 22,000 training data points (case 2).



(a) $x = 0.30$ m (b) $x = 0.50$ m

(c) $x = 0.75$ m (d) $x = 1.00$ m

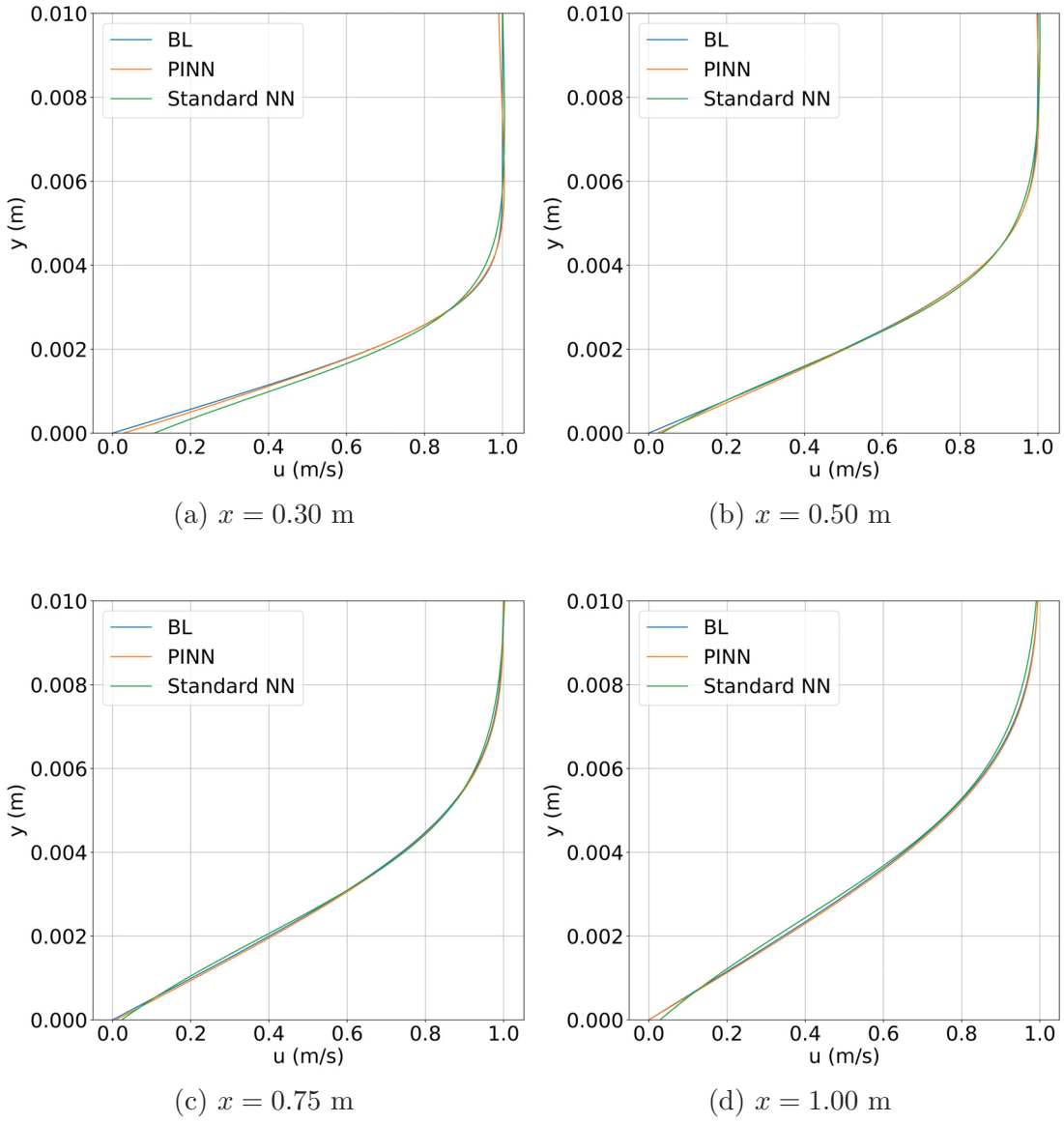
The streamwise velocity component profiles for Case 3, with 11,000 training data

points, are also examined in Fig. 10. For the first streamwise position at $x = 0.30$ m, shown in Fig. 10a, the PINN clearly demonstrates higher accuracy compared to the prediction made by the SNN at this location, particularly in the near-wall region. This behavior is consistent with the previous cases, where the standard neural network fails to provide an accurate prediction near the boundary layer edge. For the positions $x = 0.50$ m and $x = 0.75$ m, illustrated in Figs. 10b and 10c, respectively, both networks deliver similar results, with good agreement with the reference solution in both the near-wall and boundary layer edge regions. This indicates that, when sufficient data is available, the SNN can achieve reasonable accuracy; however, the PINN continues to match or slightly improve upon these results. At the last position, $x = 1.00$ m (Fig. 10d), the standard neural network once again exhibits lower accuracy close to the wall. A small deviation is also observed at the boundary layer edge, reinforcing the advantage of the PINN in these scenarios. Overall, these findings emphasize that the PINN maintains superior accuracy, especially in cases with reduced data availability for training. This robustness highlights the capability of PINNs to generalize more effectively than standard neural networks, ensuring better prediction quality even under sparse-data conditions.

Streamwise velocity component contours, as predicted by the boundary layer solver and both standard and physics-informed neural networks, are depicted in Fig. 11. For this investigation, our flowfield predictions correspond to streamwise positions ranging from $x = 0.25$ m to $x = 1.00$ m. Because the training dataset extends from $x = 0.50$ m to $x = 1.00$ m, these predictions include regions that were not used to train the neural networks. We observe that, for our three training datasets, whose dimensionality spans from 11,000 to 44,000 points, the PINN provides a superior estimation of the velocity field, closely reproducing the results generated with the boundary layer flow solver. Conversely, the results from the standard neural network are less satisfactory for the same training data sets, showing noticeable deviations, particularly in regions with higher velocity gradients. For cases 2 and 3, which consider reduced training data sets, the boundary layer edge presents a linear evolution in the standard neural network predictions, which does not represent the true physics. The PINN results, on the other hand, indicate a boundary layer edge that retains the boundary layer flow features and compare favorably with the results obtained with the boundary layer flow solver. This represents the physics-informed approach's ability to generalize from a limited training dataset.

The present results underscore the superiority of the PINN in terms of accuracy and adherence to physical constraints, highlighting its potential application in simulations where the fidelity of physical models is crucial. Additionally, once trained, the PINN model can be easily coupled with other flow solvers, which results in less complex and more effective flow simulation frameworks.

Figure 10 – Streamwise velocity component, u, profiles as predicted by the boundary layer solver and both standard and physics-informed neural networks for for 11,000 training data points (case 3).



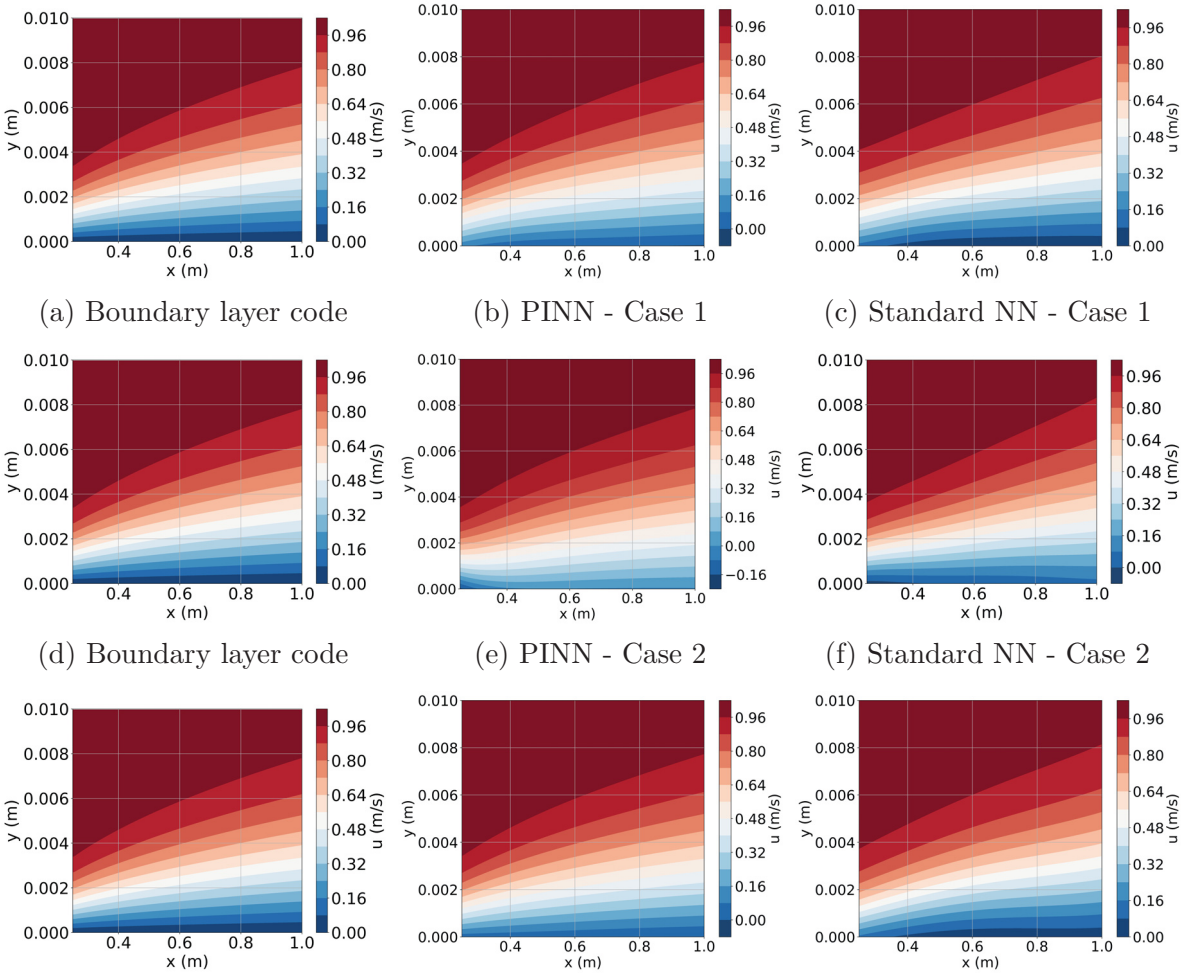
(a) $x = 0.30$ m (b) $x = 0.50$ m

(c) $x = 0.75$ m (d) $x = 1.00$ m

6.2 Boundary layer flow prediction with TensorFlow

Due to certain limitations encountered while using SciANN, particularly regarding the flexibility to modify network architecture, loss functions, and other custom features required for the PSE problem, we implement a custom Physics-Informed Neural Network code using TensorFlow 2. The SciANN library, while convenient for standard PINN implementations, presented challenges when adapting the code to specific features without modifying its internal structure, which could risk breaking the library's functionality. By using TensorFlow 2, we gained greater control and customization capabilities, allowing us to directly adjust the network architecture, tailor the loss function to our problem, and implement additional features as needed. Furthermore, this implementation facilitated

Figure 11 – Streamwise velocity, u, contours as predicted by the boundary layer solver and both standard and physics-informed neural networks, for all training data sets considered here, obtained using SciANN.



(a) Boundary layer code (b) PINN - Case 1 (c) Standard NN - Case 1

(d) Boundary layer code (e) PINN - Case 2 (f) Standard NN - Case 2

(g) Boundary layer code (h) PINN - Case 3 (i) Standard NN - Case 3

systematic experiments, such as comparing prediction accuracy across different training data volumes and evaluating the mean squared error (MSE) of the PINN predictions. In the following sections, we demonstrate that the TensorFlow implementation achieves results that are at least as accurate as those obtained with SciANN, while providing enhanced flexibility and maintainability for further modifications.

6.2.1 Data generation

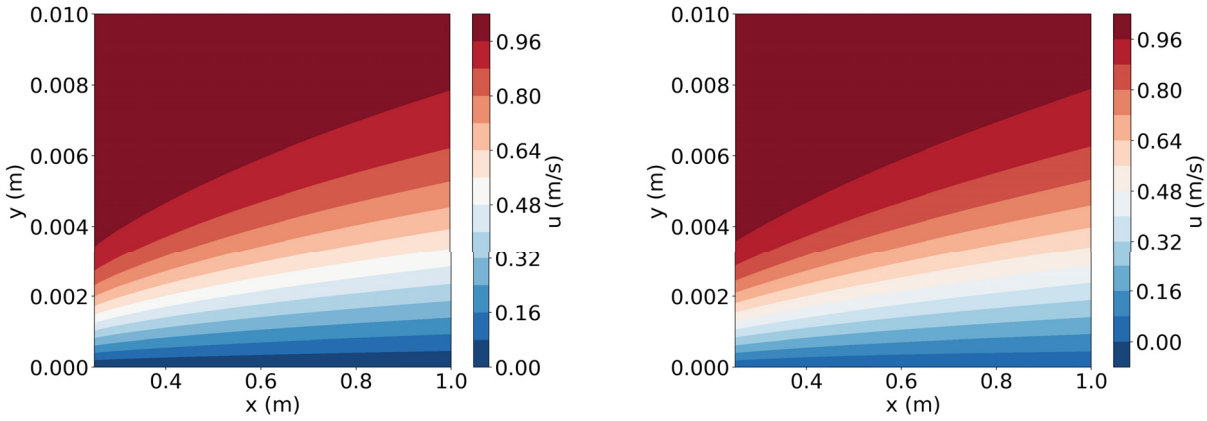
For our PINN implementation, the training dataset is obtained by running the boundary layer code detailed in Sec. 3.2. We use a computational domain with 100 points in both x- and y-directions. This discretization generates a dataset consisting of 10,000 points for each velocity component, namely u and v. The present investigation focuses on flat plate boundary layers under conditions of zero pressure gradient. The freestream velocity is defined as $U_\infty = 1.00$ m/s, and the flow is considered laminar, with a Reynolds

number of 250,000 based on the length of the flat plate. The computational domain spans from $x = 0.25$ to $x = 1.00$ m. Although the v-velocity component values were significantly smaller than the u-velocity component, as expected in boundary layer flows, no scaling was applied as it proved unnecessary. Therefore, the total training dataset consists of 20,000 points, with 60% of the data points allocated for training the neural network, corresponding to 12,000 data points. The remaining 8,000 points were reserved for model testing.

6.2.2 Results: boundary layer flow prediction with TensorFlow

For the TensorFlow implementation, the network is configured with an architecture consisting of 8 hidden layers, each containing 40 neurons, and trained using a learning rate of 0.001. In Fig. 12, the streamwise velocity component flow field indicates how well the PINN predicts both the magnitudes of velocity and the boundary layer topology. However, in the initial portion of the domain, one can observe a slight discrepancy in magnitude.

Figure 12 – Streamwise velocity component, u, contours as predicted by the boundary layer code (a) and by the physics-informed neural network (b) with TensorFlow.

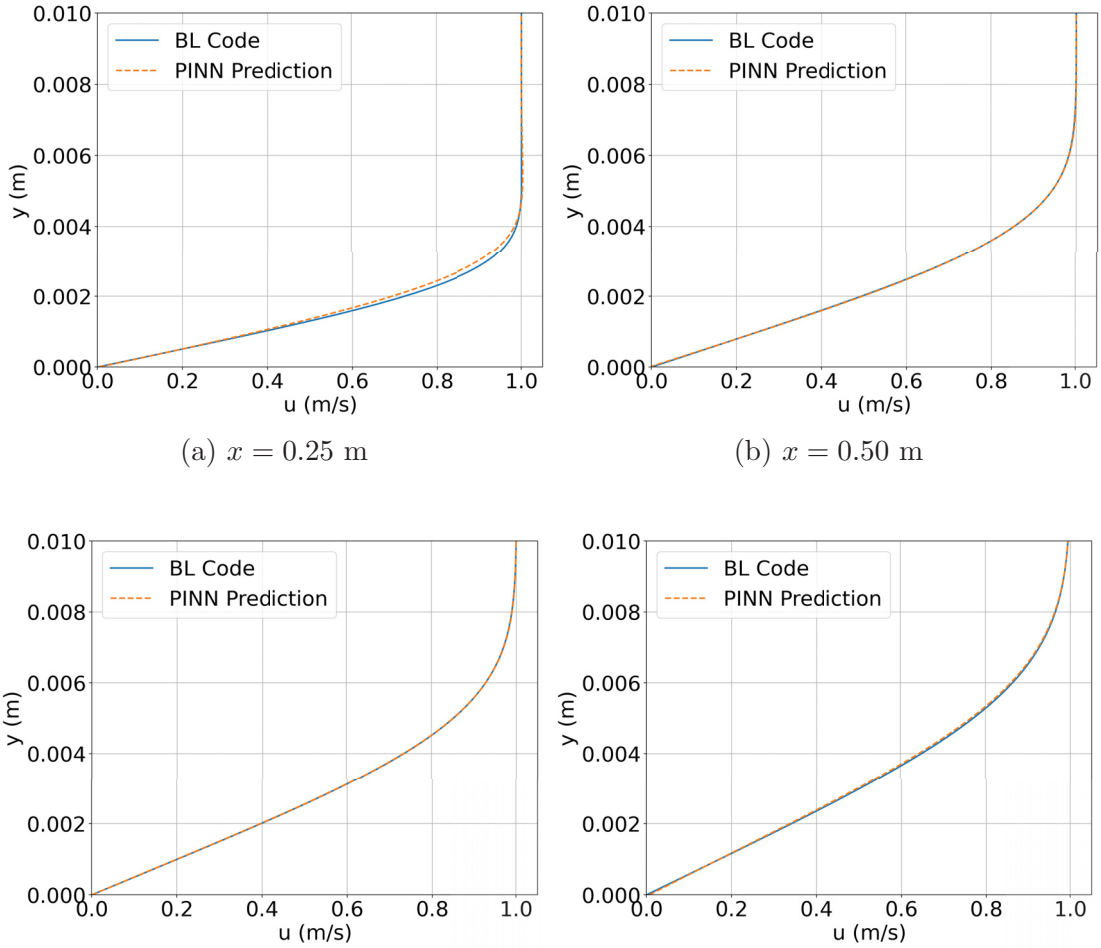


(a) Boundary Layer code

(b) PINN

This discrepancy in the initial flat plate positions can be more thoroughly analyzed in Fig. 13a, which highlights the regions with the least accurate flow field reproductions. The observed differences in the velocity profile at $x = 0.25$ m arise from the characteristics of the boundary layer code, which utilizes a marching technique due to the parabolic nature of the boundary layer equations. To initialize the boundary layer solution, the first two stations are necessary because the marching scheme relies on a second-order accurate method. The solution at these initial locations is derived from the Blasius similarity solution, presented in Sec. 3.2.1. As the marching progresses in the x-direction, the errors originating from the initial two stations estimation gradually diminish, leading to a more accurate solution. Consequently, the predictions from the PINN and the results from the boundary layer code converge as x approaches 1.00 m. By evaluating the test dataset

Figure 13 – Streamwise velocity component, u, profiles as predicted by the boundary layer solver and physics-informed neural networks with TensorFlow.



(a) $x = 0.25$ m (b) $x = 0.50$ m

(c) $x = 0.75$ m (d) $x = 1.00$ m

against the model's predictions, we calculate the mean squared error (MSE) for both the u and v velocity components. The MSE for the streamwise velocity component, u, is computed as 5.5818×10^{-6}, whereas for the wall-normal velocity component, v, the MSE is 2.6271×10^{-7}. These findings highlight the high accuracy of the PINN-based model in predicting the boundary layer flow field, particularly for the wall-normal velocity component, which exhibits significantly lower errors. The small magnitude of the MSE values further confirms that the model successfully captures the essential characteristics of the boundary layer flow.

6.3 Flow stability analysis results based on the parabolized stability equations

In this section, physics-informed neural networks, implemented using the TensorFlow framework, are trained to recover the physical behavior of spatially-evolving flow

modes that are typical in boundary layer flow stability investigations. To recover boundary layer-like behavior, focus is directed toward nonparallel, nonlocal stability analysis based on the parabolized stability equations (PSE).

6.3.1 Data generation for flow stability analysis

The training dataset is obtained from the PSE code generated with 80 points in the x-direction and 250 points, which are clustered near the wall, in the y-direction, forming a training dataset of 20,000 data points for each perturbation component, $\hat{u}$, $\hat{v}$, $\hat{w}$, and $\hat{p}$. More details on the incompressible PSE code can be found in Ref. [5]. We also predict the streamwise wave number, α. The streamwise wavenumber represents a prediction challenge for both SNN and PINN approaches. We are currently focusing on enhancing the neural network's ability to accurately recover the streamwise wavenumber spatial evolution, as originally predicted by the PSE solver. We focus on disturbances aligned with the streamwise direction ($\beta = 0$), thereby disregarding the spanwise disturbance velocity component, $\hat{w}$, which is, in this case, zero. The flow conditions are characterized by the local Reynolds number, which is based on the characteristic length scale, δ, at the first and last x-stations, with values of 400 and 1,000, respectively. The base flow represents a zero pressure gradient flat plate boundary layer flow.

6.3.2 Results of PINNs for the PSE

In this section, numerical results for the PINN-based disturbance field reconstruction are presented. The PINN-based predictions are compared with those obtained by using a standard neural network. The architecture and training parameters of the PINN code are designed to ensure accurate learning while minimizing computational cost. The ADAM optimizer, presented in Sec. 2.4, a widely used first-order stochastic gradient descent method, is employed to minimize the loss function. To approximate the reference solution, an architecture consisting of 5 hidden layers with 100 neurons each was employed. Compared to other tested configurations, this specific architecture yielded better predictive performance without significantly increasing training time. The hyperbolic tangent activation function is used, and the learning rate is set to present an exponential decay, starting from 0.001.

Initial results are obtained considering a sparse training set of 1,000 data points generated by the PSE solver. This dataset is supplemented with 20,000 collocation points, which are used to enforce the PDE residuals within the loss function. The total loss function, shown in Fig. 14, displays an error reduction as the number of epochs increases, with the maximum set at 100,000 epochs. The loss function in Fig. 14 consists of the first term of Eq. (5.8), where the neural network adjusts its output to the input data (observed data), and the second term, which is related to the parabolized stability equations.

Figure 14 – PINN loss function.

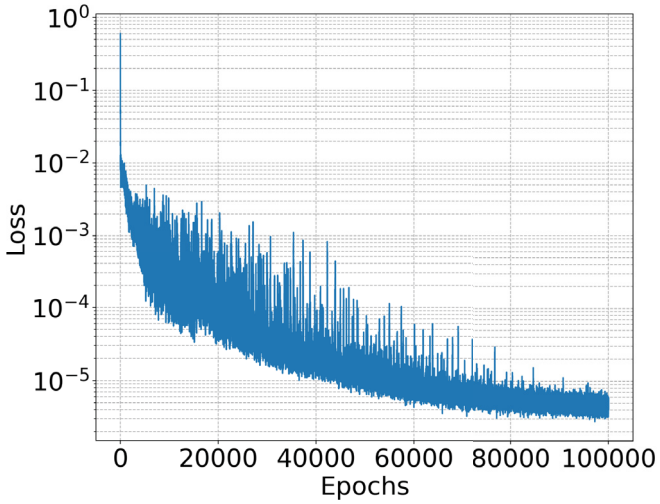
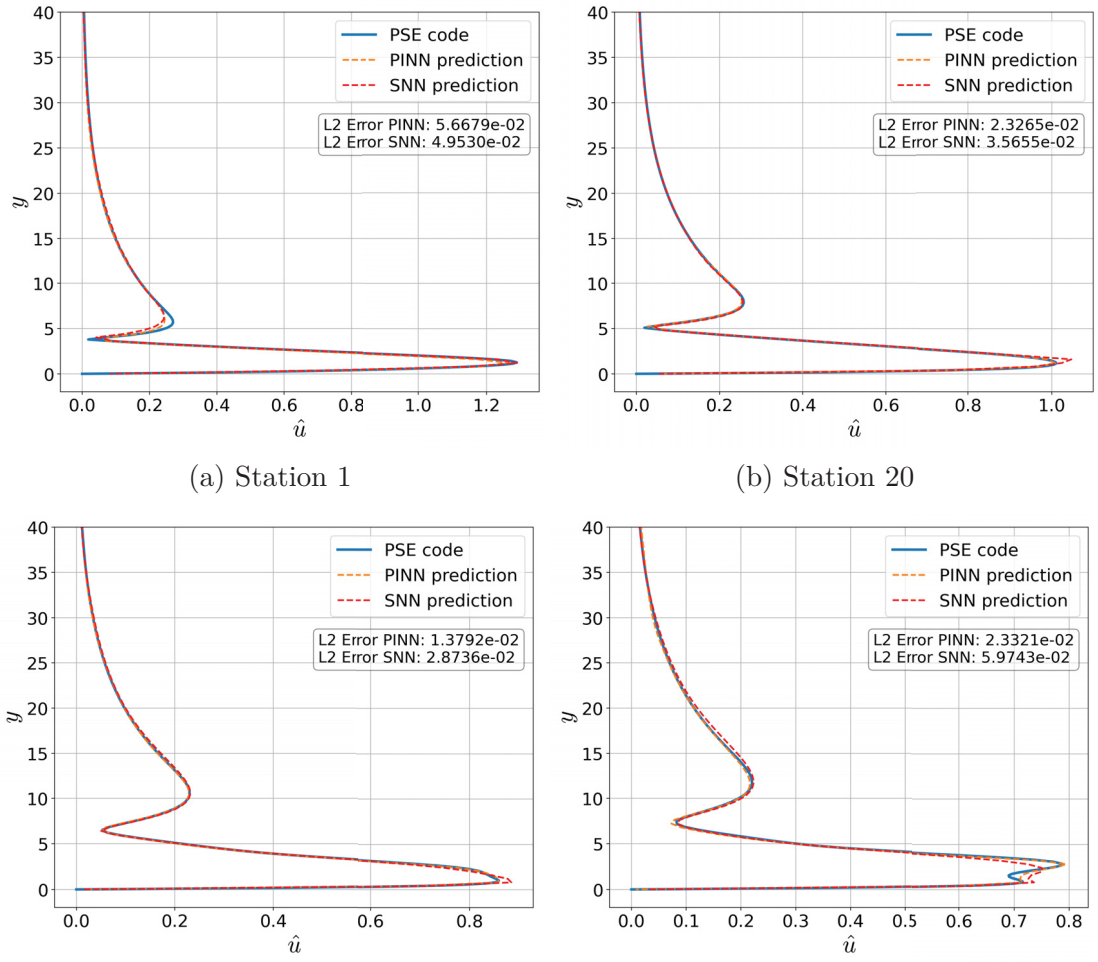


Figure 15 illustrates the comparison between the results obtained with the PINN and a standard neural network (SNN). The standard neural network was trained using the same number of points as the PINN, 1,000 points. It can be observed that the PINN shows a better fit to the PSE data. Indeed, inspection of Fig. 15 indicates that the relative L^2 norm error for streamwise stations 20, 60, and 80 is smaller for the PINN when compared to the SNN. The SNN's limitation is most apparent at station 80, which is located at the end of domain, where it is unable to accurately capture the solution's complex behavior within the region of maximum amplitude, failing to reproduce the multiple inflection points present in the true solution.

For the $\hat{v}$-eigenfunction, displayed in Fig. 16, both the PINN and the SNN yield comparable predictions. However, a closer analysis of the relative error reveals that the SNN performs better at earlier stations (1 and 20), whereas the PINN outperforms the SNN at downstream stations (60 and 80). When investigating the pressure eigenfuncion, $\hat{p}$, one can see that, indeed, the PINN outperforms the SNN. As illustrated in Fig. 17, the PINN consistently surpasses the SNN by yielding lower relative errors and more accurate near-wall predictions. The SNN presents difficulties in recovering the eigenfunction smoothness close to the wall in all streamwise flow stations, even in the one in which its relative error reaches its lowest value.

The following results assess the differences in predictions made by both PINN and SNN of for a training performed under a more data-sparse condition. For this case, both models are trained using only 500 data points generated with the PSE solver. To evaluate the PINN performance with varying levels of physical information, we test two scenarios using 10,000 and 30,000 collocation points, respectively. This setup allows us to assess the PINNs ability to accurately predict the solution fields when fewer data points are known but more collocation points are used to enforce the governing physics. The results in Fig. 18 show that increasing the number of collocation points significantly improves the

Figure 15 – Eigenfunction $\hat{u}$ absolute values in different flat plate streamwise stations.



(a) Station 1

(b) Station 20

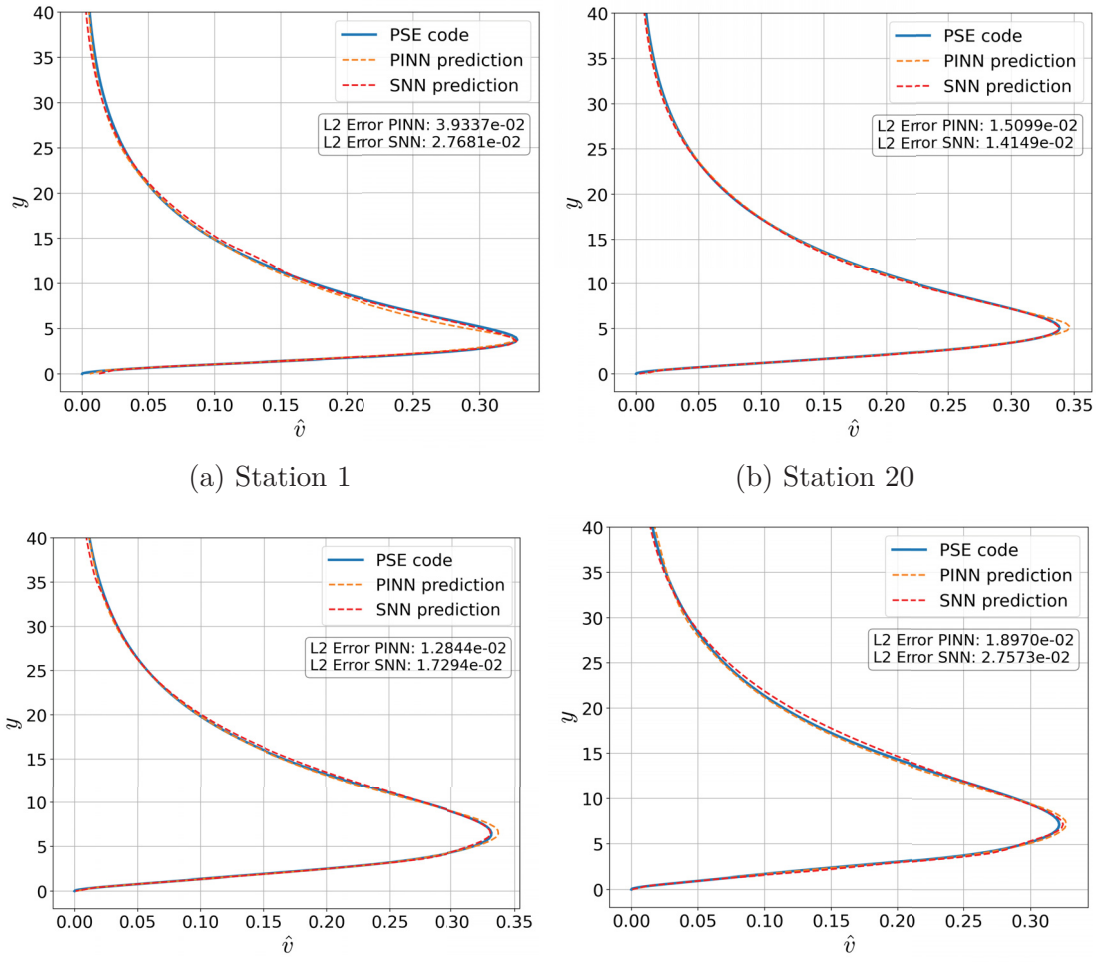
(c) Station 60

(d) Station 80

PINNs accuracy. The configuration with 500 data points and 30,000 collocation points yields more accurate predictions than both a PINN with only 10,000 collocation points and a standard neurl network trained solely on the 500 data points from the PSE solver. A visual inspection of the eigenfunction predictions confirms that the PINN provides a better fit, while the SNN produces noisier spatial amplitude distributions. This characteristic highlights how enforcing the differential equations at the collocation points acts as a strong regularizer during training, guiding the model toward a physically meaningful solution.

The analysis of the $\hat{v}$-eigenfunction, as seen in Fig. 19, further reveals the limitations of the standard neural network. The SNN's prediction deviates significantly from the true PSE solution and displays non-physical behaviors. Conversely, the PINN models provide a much better fit, an advantage that is most evident at station 80. Here, both the 30,000 and 10,000 PINN collocation point configurations demonstrate the advantages of considering the governing partial differential equations in the training. For the $\hat{p}$-eigenfunction, Fig. 20 shows that the PINN predictions are indeed more accurate than those of the SNN. The relative L^2 error values, which are presented in Fig. 20, confirm that the configuration

Figure 16 – Eigenfunction $\hat{v}$ absolute values in different flat plate streamwise stations.



(a) Station 1

(b) Station 20

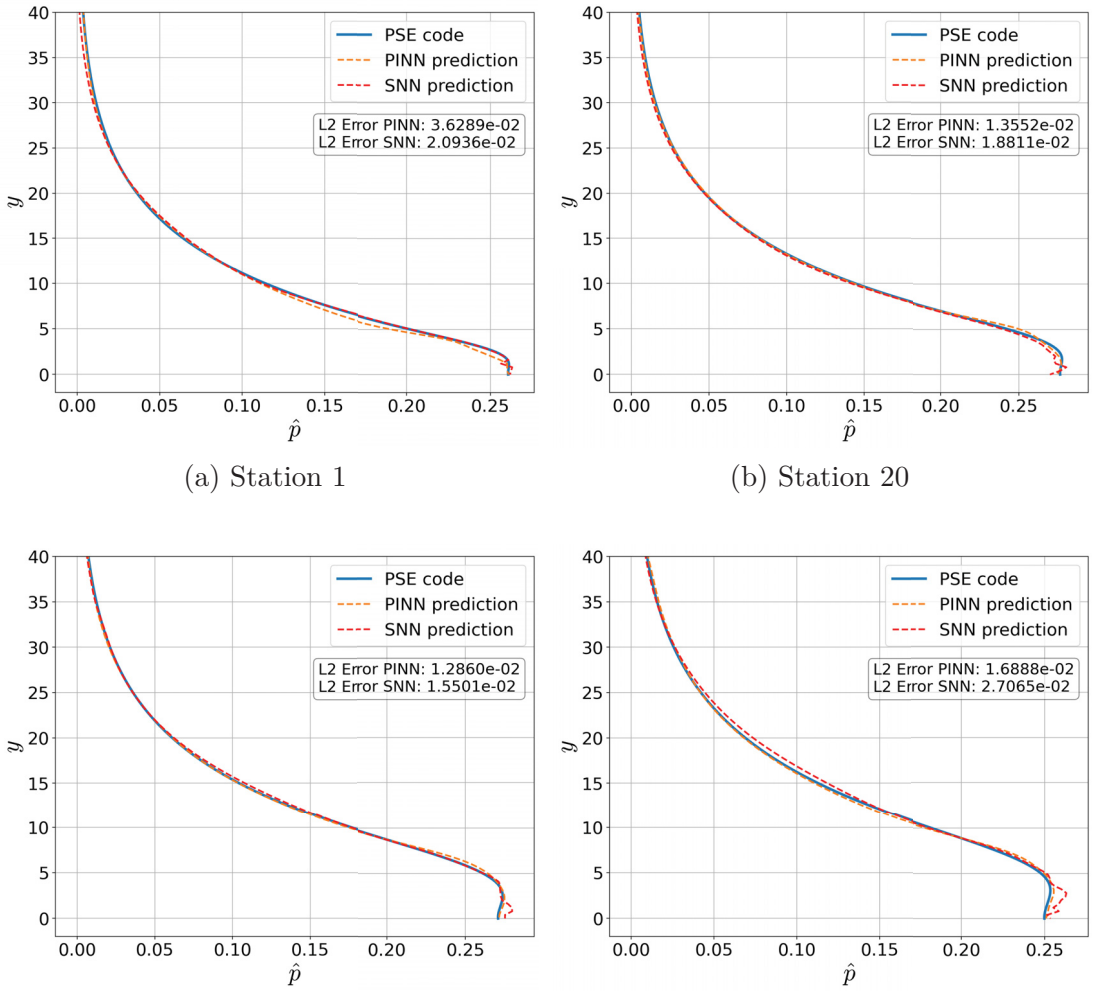
(c) Station 60

(d) Station 80

with 30,000 collocation points yields the most satisfactory results. Furthermore, the SNN struggles to predict the near-wall regions, where it exhibits non-physical oscillations that are inconsistent with the underlying physics.

Predictions of the streamwise wavenumber, α, were also carried out, allowing us to assess the accuracy of the models by analyzing both the real and imaginary parts of the wavenumber. As shown in Fig. 21, the results indicate that the PINN-based predictions are significantly more accurate, while the standard neural network exhibits a response that deviates substantially from the expected reference behavior. It can also be observed that increasing the number of collocation points leads to a reduction in the prediction error for both components of α. This highlights the importance of data density in the training process. Nevertheless, even in cases where the amount of training data is limited, the PINN is able to maintain a high level of accuracy. This is primarily due to the incorporation of the governing equations of the problem into its training, which provides additional physical constraints and compensates for the data sparsity. These results emphasize the inherent robustness of PINNs compared to conventional neural networks. While standard networks

Figure 17 – Eigenfunction $\hat{p}$ absolute values in different flat plate streamwise stations.



(a) Station 1

(b) Station 20

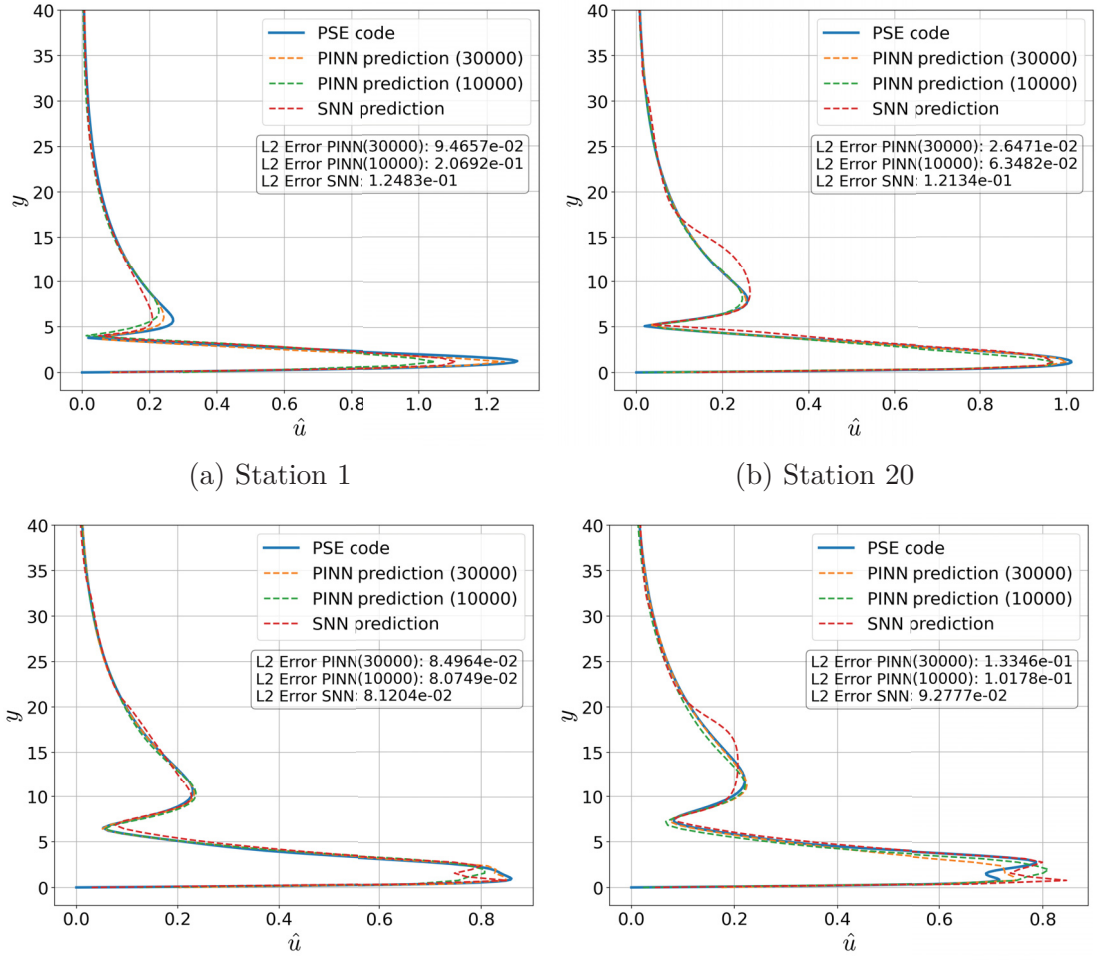
(c) Station 60

(d) Station 80

strongly depend on the availability of large amounts of data to achieve reliable predictions, PINNs are capable of delivering accurate results even under sparse-data conditions, making them a powerful tool for flow stability analysis.

Based on the previously obtained results for the disturbance fields and the streamwise wavenumber, α, it is possible to evaluate key transition prediction metrics, such as the growth rate based on kinetic energy and the N-factor, as introduced in Sec. 4.1.2. Figure 22 shows the energy-based growth rate predictions provided by each network in comparison with the reference results obtained from the PSE code. The curve predicted by the standard neural network (SNN) proves insufficient for analyzing this metric when trained with only 500 points, whereas the PINN yields a prediction much closer to the reference curve, using the same number of training points. By analyzing the relative L^2 error for both curves, it becomes clear that the SNN produces inadequate predictions, with an error of 2.9360×10^{-1}, while the PINN achieves a significantly lower error of 3.6790×10^{-2}. For the N-factor, as shown in Fig. 23, the inability of the SNN to provide satisfactory results under sparse-data conditions is again evident. In contrast, the PINN

Figure 18 – Eigenfunction $\hat{u}$ absolute values for 500 data points in different flat plate streamwise stations.



(a) Station 1

(b) Station 20

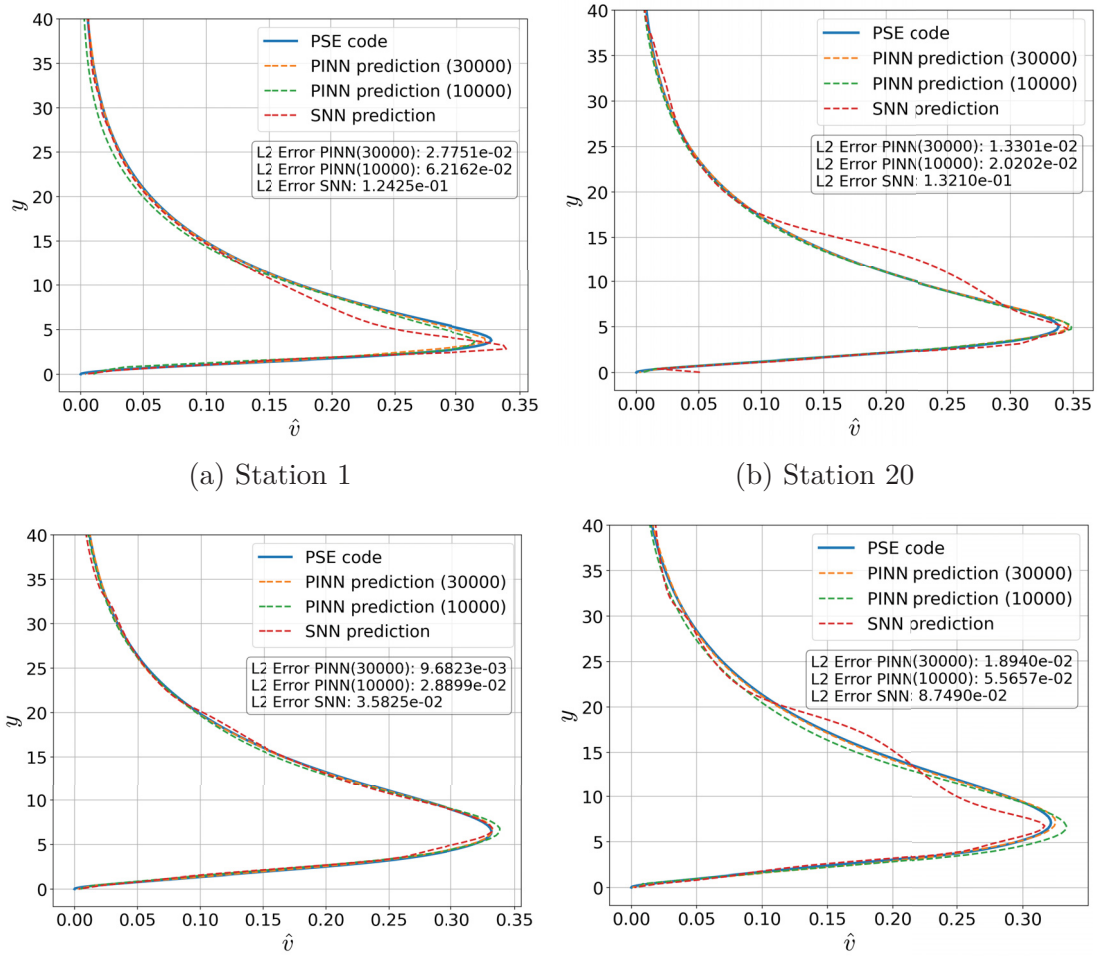
(c) Station 60

(d) Station 80

delivers a much more accurate prediction, proving effective in sparse-data scenarios and suitable for transition-to-turbulence prediction. The superiority of the PINN over the SNN becomes even more evident when examining the relative errors associated with these curves: the PINN achieves a relative error of 3.9036×10^{-2}, whereas the SNN yields a considerably larger error of 3.4581×10^{-1}.

Assessing the PINN eigenfunction predictions against the disturbance field generated by the PSE solver is essential for understanding and improving the machine learning model. This analysis allows the evaluation and adjustment of model parameters, aiming at progressively more accurate predictions. In this context, PINNs emerge as an important approach for the analysis of complex fluid flow phenomena, such as transition and turbulence. The current improved eigenfunction predictions, when compared against those obtained with standard neural networks, combined with a reduced computational cost, render PINNs a suitable approach to investigate flow stability problems. The choice of the number of collocation points in a PINN architecture represents a critical trade-off

Figure 19 – Eigenfunction $\hat{v}$ absolute values for 500 data points in different flat plate streamwise stations.



(a) Station 1

(b) Station 20

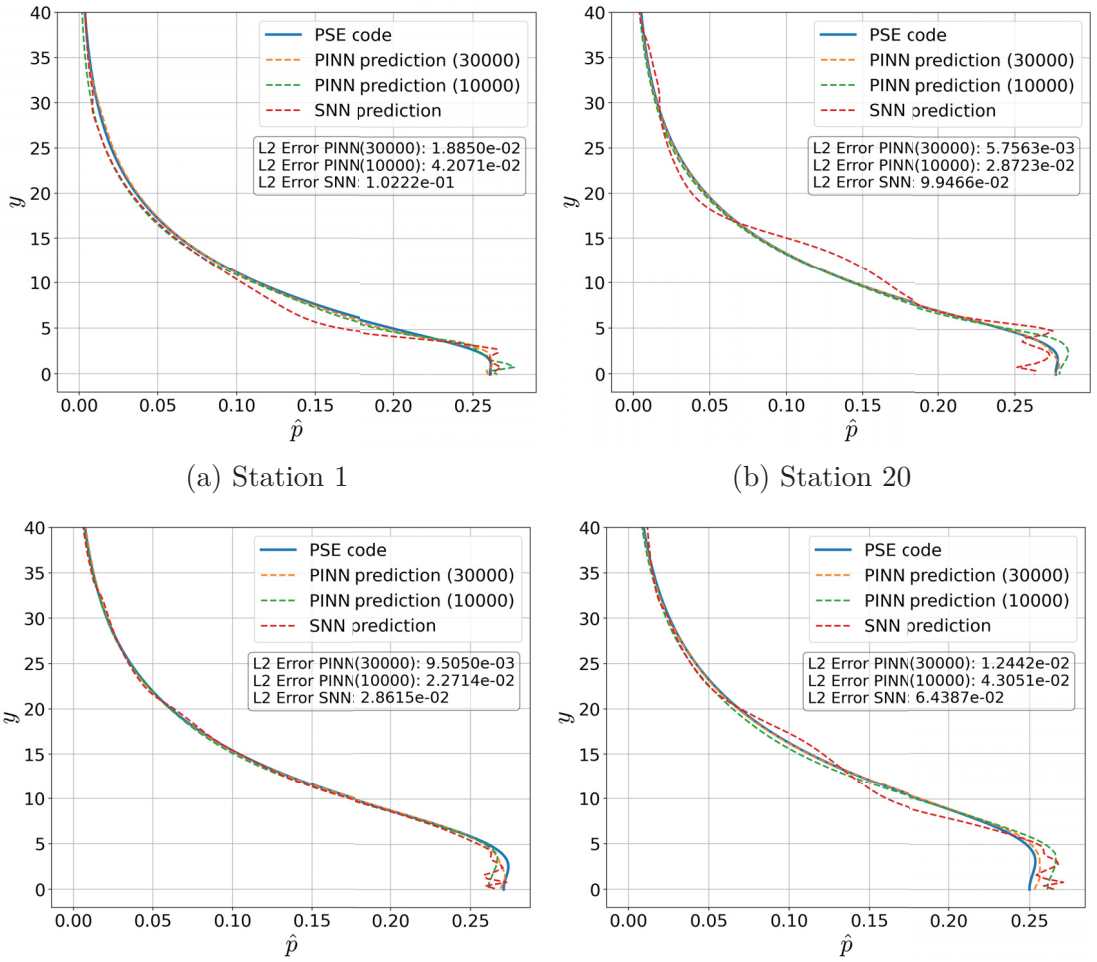
(c) Station 60

(d) Station 80

between accuracy and computational cost.

The network's training time is directly proportional to the number of points used, since each additional point requires the calculation of the PDE residual and the subsequent backpropagation of its gradients. Increasing the number of collocation points from 10,000 to 30,000 results in an increase in training time from 298 to 352 minutes on a machine equipped with a ninth-generation Intel i5-9400F processor, featuring 6 cores and 6 threads, operating at a frequency of 4.10 GHz. The system also has 16 GB of DDR4-2666 RAM. Additionally, the PINN's ease of operation, when contrasted with full-order flow solvers, which require in-depth operational knowledge, equips the researcher and the practitioner alike with a powerful analysis tool.

Figure 20 – Eigenfunction $\hat{p}$ absolute values in different flat plate streamwise stations.



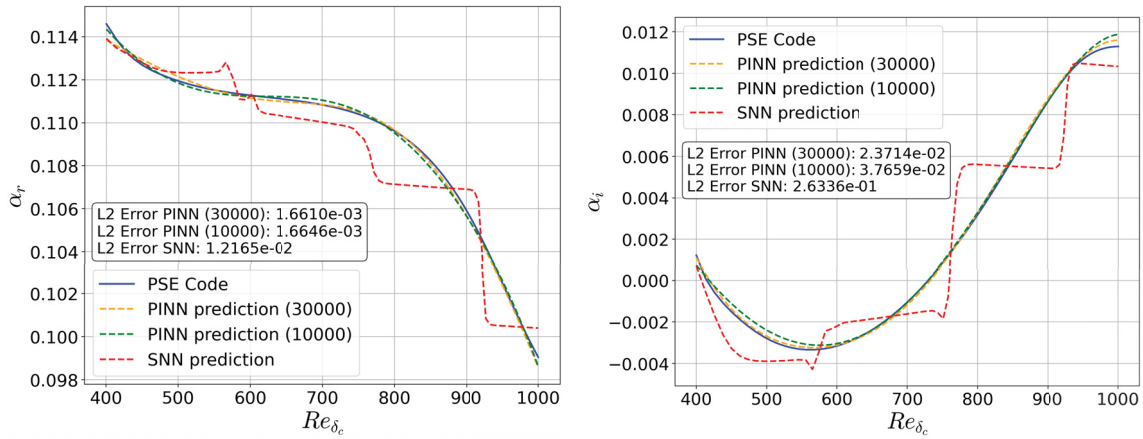
(a) Station 1

(b) Station 20

(c) Station 60

(d) Station 80

Figure 21 – Real and imaginary streamwise wavenumber parts as predicted by neural networks and the PSE code.



(a) Real component.

(b) Imaginary component.

Figure 22 – Energy-based growth rates as predicted by neural networks and the PSE code.

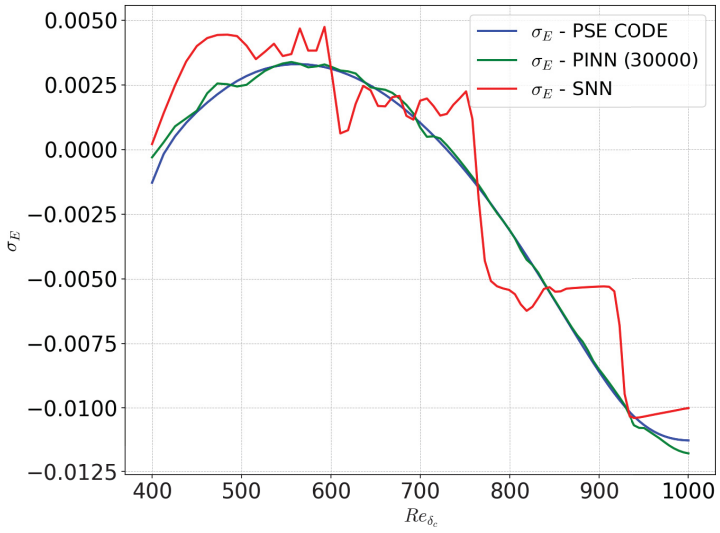
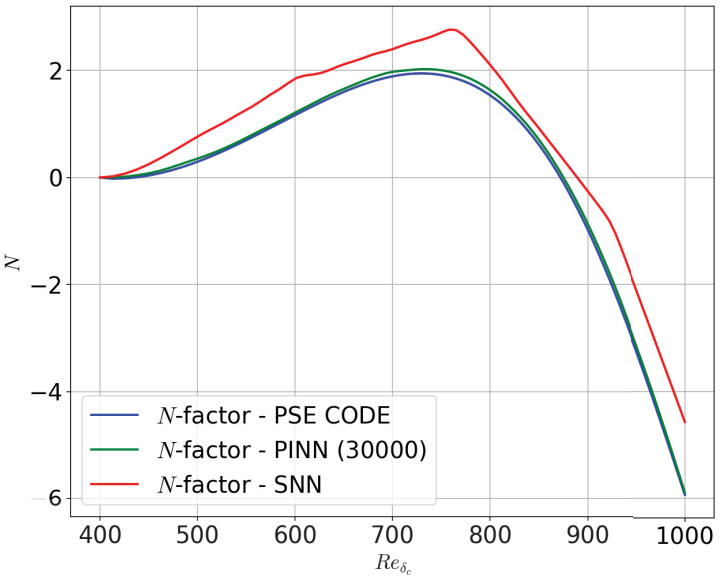


Figure 23 – N-factor as predicted by neural networks and the PSE code.



7 Conclusions

This work explored physics-informed, data-driven strategies for modeling laminar boundary layer flows and for the reconstruction of disturbance fields relevant to flow stability analysis. A numerical framework for generating reference datasets, including baseflow solutions from a boundary layer solver and disturbance fields obtained with a PSE-based solver, was developed and used to train and evaluate both standard neural networks (SNNs) and physics-informed neural networks (PINNs).

7.1 Final remarks

The results reported in Chapter 6 indicate that PINNs deliver improved accuracy in reproducing velocity profiles and boundary layer metrics when compared with conventional data-driven models, while presenting a favorable trade-off between data requirements and model accuracy. The numerical results presented here also show that PINNs are capable of reconstructing both eigenfunction and streamwise wavenumber spatial evolutions even when only sparse training data are available. Taken together, these outcomes demonstrate the potential of PINNs as automated surrogate tools that can complement classical CFD and stability solvers in studies of wall-bounded flows.

Regarding the main technical contributions and lessons learned, a modular PINN architecture was developed and trained using collocation strategies tailored to the boundary layer problem, supported by a consistent procedure to generate training and test datasets from the reference solvers, thus enabling fair comparisons between PINNs and SNNs. For the flat plate case analyzed here, the PINNs consistently recovered velocity profiles and the spatial evolution of boundary layer thickness with higher accuracy than SNNs, particularly in near-wall regions and under reduced-data scenarios. The results also reveal that PINNs can reconstruct eigenfunction-like disturbance fields and capture wave-like features characteristic of Tollmien–Schlichting wave dynamics, suggesting a practical pathway for integrating data-driven models with modal stability analysis. Furthermore, the study shows that the number of collocation points and the choice of PINN hyperparameters have a critical impact: while larger collocation sets improve accuracy, they substantially increase training time, underscoring the importance of adaptive sampling strategies and computational acceleration for extending PINN applicability to larger-scale problems. In addition, the strong ability of PINNs to reproduce disturbance fields and streamwise wavenumbers led to markedly more accurate estimates of key transition metrics such as the growth rate and the N-factor, reinforcing their suitability for reliable prediction of transition regions.

The superiority of the physics-informed approach over standard neural networks was quantitatively demonstrated, especially in data-scarce scenarios. In the stability analysis using only 500 training points, the PINN reduced the prediction error of the disturbance growth rate (σ_E) from 2.936×10^{-1} (SNN) to 3.679×10^{-2}, an improvement of approximately eight times. Similarly, for the computation of the N-factor, which is crucial for predicting the laminar-turbulent transition, the PINN presented a relative error of only 3.90×10^{-2}, in contrast to the error of 3.45×10^{-1} obtained by the SNN. Regarding the boundary layer prediction, the PINN demonstrated greater robustness, maintaining errors for the streamwise velocity component (u) below 0.71% even when the training dataset was reduced by half (22,000 points), whereas the SNN exhibited a performance degradation exceeding 1%.

Despite the promising results, certain limitations were identified. The current PINN implementations require careful hyperparameter tuning, including network size, collocation density, and loss weighting—to prevent nonphysical oscillations and ensure stable training. Moreover, the present analysis is restricted to two-dimensional, laminar boundary layer configurations over a flat plate, and further studies are needed to extend the approach to three-dimensional, curved, or strongly nonparallel flows.

Overall, this work demonstrates that physics-informed machine learning methods offer a promising route to reducing the computational and data demands associated with baseflow reconstruction and preliminary stability analyses. By combining physical constraints with the flexibility of data-driven models, PINNs provide a framework capable of accelerating the development of surrogate models for engineering applications while preserving interpretability through explicit PDE residuals and boundary conditions.

7.2 Future work

Based on the results obtained so far, several additional aspects will be investigated in future work, with the aim of enhancing the generalization and robustness of the approach. A first essential step is to complete the coupling between the PINN-generated baseflows and the PSE stability solver, creating an automated interface that allows the disturbance growth rates and neutral point locations predicted by the PINN–PSE workflow to be validated against the reference PSE solution and, when available, against DNS or experimental data. Following this, a systematic parametric study across a wide range of Reynolds numbers should be performed to assess the robustness of the PINN formulation, identify performance regimes, and compute N-factor envelopes relevant to transition prediction.

Another important line of investigation involves implementing adaptive collocation strategies, such as error-based refinement or active learning, to concentrate collocation

points in regions where the PDE residual is largest, typically near the wall or within shear layers. This approach can reduce overall training cost while maintaining the desired accuracy. In parallel, incorporating uncertainty quantification through Bayesian PINNs or ensemble methods will be crucial to estimate predictive uncertainty; propagating uncertainties in inputs such as inflow profiles or viscosity will allow confidence intervals to be assigned to predicted growth rates.

Hybrid PINN–CFD workflows also represent a promising avenue, where PINNs can accelerate specific portions of a traditional CFD pipeline, such as near-wall resolution, while a conventional solver ensures global consistency, enabling an evaluation of potential speedups and accuracy gains. Extending the present formulation to three-dimensional boundary layers and flows over curved surfaces is likewise a necessary step to assess scalability and applicability to realistic aerodynamic configurations. Finally, a detailed investigation of normalization strategies and loss-weighting approaches tailored to flow-stability problems, such as phase-fixing or energy normalization techniques commonly used in PSE, will be essential to guarantee consistent training behavior and physically meaningful disturbance reconstructions.

Taken together, these research directions will transform the current methodological framework into a mature and versatile toolset for boundary-layer prediction and stability analysis. Prioritizing adaptive sampling, uncertainty quantification, and thorough numerical and experimental validation will be fundamental to establishing PINNs as reliable surrogate models in transition studies and aerodynamic design.

References

[1] P. G. Drazin. *Introduction to Hydrodynamic Stability*. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511809064.

[2] H. Schlichting and K. Gersten. *Boundary-Layer Theory*. Berlin: Springer, 2017. DOI: 10.1007/978-3-662-52919-5.

[3] P. Moin and K. Mahesh. "DIRECT NUMERICAL SIMULATION: A Tool in Turbulence Research". In: *Annual Review of Fluid Mechanics* 30.Volume 30, 1998 (1998), pp. 539–578. ISSN: 1545-4479. DOI: https://doi.org/10.1146/annurev.fluid.30.1.539. URL: https://www.annualreviews.org/content/journals/10.1146/annurev.fluid.30.1.539.

[4] T. Herbert. "Parabolized Stability Equations". In: *Annual Review of Fluid Mechanics* 29 (1997), pp. 245–283. DOI: 10.1146/annurev.fluid.29.1.245.

[5] M. P. Juniper, A. Hanifi, and V. Theofilis. "Modal Stability Theory. Lecture notes from the FLOW-NORDITA Summer School on Advanced Instability Methods for Complex Flows. Stockholm, Sweden, 2013". In: *Aplied Mechanics Reviews* 66 (Mar. 2014), pp. 1–22. DOI: 10.1115/1.4026604.

[6] G. L. O. Halila, K. J. Fidkowski, and J. R. R. A. Martins. "Toward Automatic Parabolized Stability Equation-Based Transition-to-Turbulence Prediction for Aerodynamic Flows". In: *AIAA Journal* 59.2 (Feb. 2021), pp. 462–473. DOI: 10.2514/1.J059516.

[7] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. "Machine Learning for Fluid Mechanics". In: *Annual Review of Fluid Mechanics* 52 (2020), pp. 477–508. DOI: 10.1146/annurev-fluid-010719-060214.

[8] J. Li, X. Du, and J. R. Martins. "Machine Learning in Aerodynamic Shape Optimization". In: *Progress in Aerospace Sciences* 134 (2022), p. 100849. DOI: 10.1016/j.paerosci.2022.100849.

[9] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen. "Recent advances in convolutional neural networks". In: *Pattern Recognition* 77 (2018), pp. 354–377. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.10.013. URL: https://www.sciencedirect.com/science/article/pii/S0031320317304120.

[10] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Machine Learning of Linear Differential Equations using Gaussian Processes". In: *Journal of Computational Physics* 348 (2017), pp. 683–693. DOI: 10.1016/j.jcp.2017.07.050.

[11] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics Informed Deep Learning (Part I): Data-Driven Solutions of Nonlinear Partial Differential Equations". In: *arXiv preprint arXiv:1711.10561* (2017).

[12] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics Informed Deep Learning (Part II): Data-Driven Discovery of Nonlinear Partial Differential Equations". In: *arXiv preprint arXiv:1711.10566* (2017).

[13] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. DOI: 10.1016/j.jcp.2018.10.045.

[14] M. Raissi, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis. "Deep Learning of Vortex-Induced Vibrations". In: *Journal of Fluid Mechanics* 861 (2019), pp. 119–137. DOI: 10.1017/jfm.2018.872.

[15] M. Raissi, P. Perdikaris, and G. E. Karniadakis. *Physics Informed Learning Machine*. US Patent 10,963,540. Mar. 2021.

[16] Q. Guo, Y. Zhao, C. Lu, and J. Luo. "High-Dimensional Inverse Modeling of Hydraulic Tomography by Physics Informed Neural Network (HT-PINN)". In: *Journal of Hydrology* 616 (2023), p. 128828. DOI: 10.1016/j.jhydrol.2022.128828.

[17] J. Han, A. Jentzen, and W. E. "Solving High-Dimensional Partial Differential Equations using Deep Learning". In: *Proceedings of the National Academy of Sciences* 115.34 (2018), pp. 8505–8510. DOI: 10.1073/pnas.1718942115.

[18] W.-X. Yuan, R. Guo, and Y.-N. Gao. "Physics-Informed Neural Network Method for the Modified Nonlinear Schrödinger Equation". In: *Optik* 279 (2023), p. 170739. DOI: 10.1016/j.ijleo.2023.170739.

[19] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. "Scientific Machine Learning Through Physics–Informed Neural Networks: Where We Are and What's Next". In: *Journal of Scientific Computing* 92.3 (2022), p. 88. DOI: 10.1007/s10915-022-01939-z.

[20] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. "Physics-Informed Machine Learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440. DOI: 10.1038/s42254-021-00314-5.

[21] W. Peng, Y. Zhang, E. Laurendeau, and M. C. Desmarais. "Learning Aerodynamics with Neural Network". In: *Scientific Reports* 12.1 (2022), p. 6779. DOI: 10.1038/s41598-022-10737-4.

[22] J. L. Hess. *Calculation of Potential Flow about Arbitrary Three-Dimensional Lifting Bodies*. MD Report J5679-01. Long Beach, CA: Douglas Aircraft Co., Oct. 1972.

[23] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. "Variational physics-informed neural networks for solving partial differential equations". In: *arXiv preprint arXiv:1912.00873* (2019). DOI: https://doi.org/10.48550/arXiv.1912.00873.

[24] O. Sallam and M. Fürth. "On the use of Fourier Features-Physics Informed Neural Networks (FF-PINN) for forward and inverse fluid mechanics problems". In: *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment* 237.4 (2023), pp. 846–866. DOI: 10.1177/14750902231166424. eprint: https://doi.org/10.1177/14750902231166424. URL: https://doi.org/10.1177/14750902231166424.

[25] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133.

[26] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386. DOI: https://doi.org/10.1037/h0042519.

[27] L. V. Fausett. *Fundamentals of neural networks: architectures, algorithms and applications*. Pearson Education India, 2006.

[28] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. DOI: 10.1017/9781108380690.

[29] B. Singh, S. Patel, A. Vijayvargiya, and R. Kumar. "Analyzing the impact of activation functions on the performance of the data-driven gait model". In: *Results in Engineering* 18 (2023), p. 101029. DOI: https://doi.org/10.1016/j.rineng.2023.101029.

[30] A. F. Agarap. "Deep learning using rectified linear units (relu)". In: *arXiv preprint arXiv:1803.08375* (2018). DOI: https://doi.org/10.48550/arXiv.1803.08375.

[31] S. Elfwing, E. Uchibe, and K. Doya. "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning". In: *Neural networks* 107 (2018), pp. 3–11. DOI: https://doi.org/10.1016/j.neunet.2017.12.012.

[32] F. Wang, J. Cheng, W. Liu, and H. Liu. "Additive Margin Softmax for Face Verification". In: *IEEE Signal Processing Letters* 25.7 (2018), pp. 926–930. DOI: 10.1109/LSP.2018.2822810.

[33] T. Li, R. Shi, and T. Kanai. "DenseGATs: A graph-attention-based network for nonlinear character deformation". In: *Symposium on Interactive 3D Graphics and Games*. 2020, pp. 1–9. DOI: https://doi.org/10.1145/3384382.3384525.

[34] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[35] S. Vani and T. V. M. Rao. "An Experimental Approach towards the Performance Assessment of Various Optimizers on Convolutional Neural Network". In: *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. 2019, pp. 331–336. DOI: 10.1109/ICOEI.2019.8862686.

[36] H. Robbins and S. Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. ISSN: 00034851. URL: http://www.jstor.org/stable/2236626 (visited on 08/31/2025).

[37] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12.null (July 2011), pp. 2121–2159. ISSN: 1532-4435. DOI: 10.5555/1953048.2021068.

[38] M. D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. DOI: 10.48550/arXiv.1212.5701. arXiv: 1212.5701 [cs.LG]. URL: https://arxiv.org/abs/1212.5701.

[39] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980* (2014). URL: https://arxiv.org/abs/1412.6980.

[40] Z. Dai, T. Li, Z.-R. Xiang, W. Zhang, and J. Zhang. "Aerodynamic multi-objective optimization on train nose shape using feedforward neural network and sample expansion strategy". In: *Engineering Applications of Computational Fluid Mechanics* 17.1 (2023), p. 2226187. DOI: 10.1080/19942060.2023.2226187.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[42] Y. Bengio, I. Goodfellow, A. Courville, et al. *Deep learning*. Vol. 1. MIT press Cambridge, MA, USA, 2017.

[43] J. R. R. A. Martins and J. T. Hwang. "Review and Unification of Methods for Computing Derivatives of Multidisciplinary Computational Models". In: *AIAA Journal* 51.11 (Nov. 2013), pp. 2582–2599. DOI: 10.2514/1.J052184.

[44] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. "Automatic Differentiation in Machine Learning: a Survey". In: *Journal of Marchine Learning Research* 18 (2018), pp. 1–43. DOI: 10.48550/arXiv.1502.05767.

[45] E. Haghighat and R. Juanes. "SciANN: A Keras/TensorFlow Wrapper for Scientific Computations and Physics-Informed Deep Learning using Artificial Neural Networks". In: *Computer Methods in Applied Mechanics and Engineering* 373 (2021), p. 113552. DOI: 10.1016/j.cma.2020.113552.

[46] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. "TensorFlow: A System for Large-Scale Machine Learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[47] F. Chollet. *Deep learning with Python*. simon and schuster, 2021.

[48] I. G. Currie. *Fundamental mechanics of fluids*. CRC press, 2002.

[49] F. M. White. *Viscous Fluid Flow*. New York, NY: McGraw-Hill, 1991.

[50] C. A. Fletcher. *Computational Techniques for Fluid dynamics 2: Specific Techniques for Different Flow Categories*. Springer Science & Business Media, 2012.

[51] H. Blasius. *Grenzschichten in Flüssigkeiten mit kleiner Reibung*. Druck von BG Teubner, 1907.

[52] L. H. Thomas. "Elliptic Problems in Linear Difference Equations over a Network". In: *Watson Sci. Comput. Lab. Rept., Columbia University, New York* 1 (1949), p. 71.

[53] W. S. Saric. *Introduction to Linear Stability*. Advances in Laminar Turbulent Transition Modeling. The Lecture Series of the von Karman Institute. 2008.

[54] A. M. O. Smith and N. Gamberoni. *Transition, Pressure Gradient and Stability Theory*. Douglas Aircraft Company Report ES 26388. Long Beach, CA: Douglas Aircraft Company, 1956.

[55] J. L. van Ingen. *A Suggested Semi-Empirical Method for the Calculation of the Boundary Layer Transition Region*. Univ. Delft Report VTH-74. Delft, The Nederlands: University of Delft, 1956.

[56] L. M. Mack. *Transition Prediction and Linear Stability Theory*. AGARD Report CP224. Lyngby, Denmark, May 1977.

[57] D. Arnal. *Boundary layer Transition: Predictions based on the Linear Theory*. AGARD Report AG 793. Apr. 1994.

[58] A. Hanifi, D. Henningson, S. Hein, F. Bertolotti, and M. Simen. *Linear Nonlocal Instability Analysis - the linear NOLOT code*. The Aeronautical Research Institute of Sweden Aerodynamics Department, FFA-TN 1994-54. 1994.

[59] N. M. El-Hady. "Nonparallel Instability of Supersonic and Hypersonic Boundary Layers". In: *Physics of Fluids A: Fluid Dynamics* 3 (Sept. 1991), pp. 2164–2178. DOI: 10.1063/1.857898.

[60] P. Schmid and P. Henningson. *Stability and Transition in Shear Flows*. New York: Springer, 2001. DOI: 10.1007/978-1-4613-0185-1.

[61] F. P. Bertolotti, T. Herbert, and P. R. Spalart. "Linear and Nonlinear Stability of the Blasius Boundary Layer". In: *Journal of Fluid Mechanics* 242 (1992), pp. 441–474. DOI: 10.1017/S0022112092002453.

[62] M. S. Broadhurst and S. J. Sherwin. "The Parabolised Stability Equations for 3D-flows: Implementation and Numerical Stability". In: *Appliend Numerical Mathematics* 58 (2008), pp. 1017–1029.

[63] D. Arnal, G. Casalis, and R. Houdeville. *Practical Transition Prediction Methods: Subsonic and Transonic Flows*. Advances in Laminar Turbulent Transition Modeling. The Lecture Series of the von Karman Institute. June 2008.

[64] M. Gaster. "A Note on the Relation Between Temporally-Increasing and Spatially-Increasing Disturbances in Hydrodynamic Stability". In: *Journal of Fluid Mechanics* 14.2 (Oct. 1962), pp. 222–224. DOI: 10.1017/S0022112062001184.

[65] F. P. Bertolotti and T. Herbert. "Analysis of the Linear Stability of Compressible Boundary Layers using PSE". In: *Theoretical and Computational Fluid Dynamics* 3 (1991), pp. 117–124. DOI: 10.1007/BF00271620.

[66] F. P. Bertolotti. *Turbulence and Transition Modelling. Ch. 8, Transition Modelling based on the PSE*. Netherlands: Kluwer Academic Publishers, 1996.

[67] P. Andersson, D. S. Henningson, and A. Hanifi. "On a Stabilization Procedure for the Parabolic Stability Equations". In: *Journal of Engineering Mathematics* 33 (Feb. 1998), pp. 311–332. DOI: 10.1023/A:1004367704897.

[68] L. N. Trefethen. *Spectral Methods in Matlab*. SIAM, 2000. DOI: 10.1137/1.9780898719598.

[69] M. R. Malik, T. Zang, and M. Hussaini. "A Spectral Collocation Method for the Navier–Stokes Equations". In: *Journal of Computational Physics* 61 (Oct. 1985), pp. 64–88.

[70] M. R. Malik. "Numerical Methods for Hypersonic Boundary Layer Stability". In: *Journal of Computational Physics* 86 (Feb. 1990), pp. 376–413. DOI: 0.1016/0021-9991(90)90106-B.